Trusted Platform Module Library
Part 4: Supporting Routines

Family "2.0"

Level 00 Revision 01.83

January 25, 2024

Published

Contact: admin@trustedcomputinggroup.org

TCG Published

TCG

Licenses and Notices

Copyright Licenses:

- Trusted Computing Group (TCG) grants to the user of the source code in this specification (the “Source Code”) a worldwide, irrevocable, nonexclusive, royalty free, copyright license to reproduce, create derivative works, distribute, display and perform the Source Code and derivative works thereof, and to grant others the rights granted herein.
- The TCG grants to the user of the other parts of the specification (other than the Source Code) the rights to reproduce, distribute, display, and perform the specification solely for the purpose of developing products based on such documents.

Source Code Distribution Conditions:

- Redistributions of Source Code must retain the above copyright licenses, this list of conditions and the following disclaimers.
- Redistributions in binary form must reproduce the above copyright licenses, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.

Disclaimers:

- THE COPYRIGHT LICENSES SET FORTH ABOVE DO NOT REPRESENT ANY FORM OF LICENSE OR WAIVER, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, WITH RESPECT TO PATENT RIGHTS HELD BY TCG MEMBERS (OR OTHER THIRD PARTIES) THAT MAY BE NECESSARY TO IMPLEMENT THIS SPECIFICATION OR OTHERWISE. Contact TCG Administration (admin@trustedcomputinggroup.org) for information on specification licensing rights available through TCG membership agreements.
- THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, COMPLETENESS, OR NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.
- Without limitation, TCG and its members and licensors disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Any marks and brands contained herein are the property of their respective owners.

CONTENTS

Trusted Platform Module Library
Part 4: Supporting Routines

1 Scope

Part 4 is provided in order to enable the explanation and demonstration, via the TCG's reference code (reproduced in Parts 3 and 4), of the functionality described in Parts 1, 2, and 3. The code in Parts 3 and 4 is not written or guaranteed to meet any level of conformance, nor does this specification require that a TPM meet any particular level of conformance. In some instances (e.g., firmware update), Part 4 cannot describe a compliant implementation. Therefore, an implementor of TPM 2.0 may decide to replace Part 4 code with vendor-specific code that enables compliance.

2 Terms and definitions

For the purposes of this document, the terms and definitions given in TPM 2.0 Part 1 apply.

3 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in TPM 2.0 Part 1 apply.

4 Automation

TPM 2.0 Part 2 and 3 are constructed so that they can be processed by an automated parser. For example, TPM 2.0 Part 2 can be processed to generate header file contents such as structures, typedefs, and enums. TPM 2.0 Part 3 can be processed to generate command and response marshaling and unmarshaling code.

4.1 Configuration

The TPM configuration is defined by the files TpmProfile*.h. These files can be edited in order to change the algorithms and commands supported by a TPM implementation.

4.2 Unmarshaling Code Prototype

4.2.1 Simple Types and Structures

The general form for the unmarshaling code for a simple type or a structure is:

```
TPM_RC TYPE_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size);
```

Where:

TYPE	name of the data type or structure
***target**	location in the TPM memory into which the data from ****buffer** is placed
****buffer**	location in input buffer containing the most significant octet (MSO) of ***target**
***size**	number of octets remaining in ****buffer**

When the data is successfully unmarshaled, the called routine will return TPM_RC_SUCCESS. Otherwise, it will return a Format-One response code (see TPM 2.0 Part 2).

If the data is successfully unmarshaled, ***buffer** is advanced point to the first octet of the next parameter in the input buffer and **size** is reduced by the number of octets removed from the buffer.

When the data type is a simple type, the parser will generate code that will unmarshal the underlying type and then perform checks on the type as indicated by the type definition.

When the data type is a structure, the parser will generate code that unmarshals each of the structure elements in turn and performs any additional parameter checks as indicated by the data type.

4.2.2 Union Types

When a union is defined, an extra parameter is defined for the unmarshaling code. This parameter is the selector for the type. The unmarshaling code for the union will unmarshal the type indicated by the selector.

The function prototype for a union has the form:

```
TPM_RC TYPE_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size, UINT32 selector);
```

where:

TYPE	name of the union type or structure
***target**	location in the TPM memory into which the data from ****buffer** is placed
****buffer**	location in input buffer containing the most significant octet (MSO) of ***target**
***size**	number of octets remaining in ****buffer**
selector	union selector that determines what will be unmarshaled into ***target**

4.2.3 Null Types

In some cases, the structure definition allows an optional “null” value. The “null” value allows the use of the same C type for the entity even though it does not always have the same members.

For example, the TPMI_ALG_HASH data type is used in many places. In some cases, TPM_ALG_NULL is permitted and in some cases it is not. If two different data types had to be defined, the interfaces and code would become more complex because of the number of cast operations that would be necessary. Rather than encumber the code, the “null” value is defined and the unmarshaling code is given a flag to indicate if this instance of the type accepts the “null” parameter or not. When the data type has a “null” value, the function prototype is

```
TPM_RC TYPE_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size, BOOL flag);
```

The parser detects when the type allows a “null” value and will always include **flag** in any call to unmarshal that type. **flag** TRUE indicates that null is accepted.

4.2.4 Arrays

Any data type may be included in an array. The function prototype use to unmarshal an array for a **TYPE** is

```
TPM_RC TYPE_Array_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size,INT32 count);
```

The generated code for an array uses a **count**-limited loop within which it calls the unmarshaling code for **TYPE**.

4.3 Marshaling Code Function Prototypes

4.3.1 Simple Types and Structures

The general form for the marshaling code for a simple type or a structure is:

```
UINT16 TYPE_Marshal(TYPE *source, BYTE **buffer, INT32 *size);
```

Where:

TYPE	name of the data type or structure
***source**	location in the TPM memory containing the value that is to be marshaled in to the designated buffer
****buffer**	location in the output buffer where the first octet of the **TYPE** is to be placed
***size**	number of octets remaining in ****buffer**.

If **buffer** is a NULL pointer, then no data is marshaled, but the routine will compute and return the size of the memory required to marshal the indicated type. ***size** is not changed.

If **buffer** is not a NULL pointer, data is marshaled, ***buffer** is advanced to point to the first octet of the next location in the output buffer, and the called routine will return the number of octets marshaled into ****buffer**. This occurs even if **size** is a NULL pointer. If **size** is a not NULL pointer ***size** is reduced by the number of octets placed in the buffer.

When the data type is a simple type, the parser will generate code that will marshal the underlying type. The presumption is that the TPM internal structures are consistent and correct so the marshaling code does not validate that the data placed in the buffer has a permissible value. The presumption is also that the **size** is sufficient for the source being marshaled.

When the data type is a structure, the parser will generate code that marshals each of the structure elements in turn.

4.3.2 Union Types

An extra parameter is defined for the marshaling function of a union. This parameter is the selector for the type. The marshaling code for the union will marshal the type indicated by the selector.

The function prototype for a union has the form:

```
UINT16 TYPE_Marshal(TYPE *source, BYTE **buffer, INT32 *size, UINT32 selector);
```

The parameters have a similar meaning as those in 4.2.2 but the data movement is from **source** to **buffer**.

4.3.3 Arrays

Any type may be included in an array. The function prototype use to unmarshal an array is:

```
UINT16 TYPE_Array_Marshal(TYPE *source, BYTE **buffer, INT32 *size, INT32 count);
```

The generated code for an array uses a **count**-limited loop within which it calls the marshaling code for **TYPE**.

4.4 Table-driven Marshaling

The most recent versions of the TPM code includes the option to use table-driven marshaling rather that the procedural marshaling described in previous clauses in 4.2. The structure and processing of this code is complex and is provided in the code.

4.5 Part 3 Parsing

The Command / Response tables in Part 3 are reflected in command-specific data structures used by functions in this TPM 2.0 Part 4. They are:

- **CommandAttributeData.h** -- This file contains the command attributes reported by TPM2_GetCapability.
- **CommandAttributes.h** – This file contains the definition of command attributes that are extracted by the parsing code. The file mainly exists to ensure that the parsing code and the function code are using the same attributes.
- **CommandDispatchData.h** – This file contains the data definitions for the table driven version of the command dispatcher.

Part 3 parsing also produces special function prototype files as described in **Error! Reference source not found.**.

4.6 Portability

Where reasonable, the code is written to be portable. There are a few known cases where the code is not portable. Specifically, the handling of bit fields will not always be portable. The bit fields are marshaled and unmarshaled as a simple element of the underlying type. For example, a TPMA_SESSION is defined as a bit field in an octet (BYTE). When sent on the interface a TPMA_SESSION will occupy one octet. When unmarshaled, it is unmarshaled as a UINT8. The ramifications of this are that a TPMA_SESSION will occupy the 0th octet of the structure in which it is placed regardless of the size of the structure.

Many compilers will pad a bit field to some "natural" size for the processor, often 4 octets, meaning that `sizeof(TPMA_SESSION)` would return 4 rather than 1 (the canonical size of a TPMA_SESSION).

For a little endian machine, padding of bit fields should have little consequence since the 0th octet always contains the 0th bit of the structure no matter how large the structure. However, for a big endian machine, the 0th bit will be in the highest numbered octet. When unmarshaling a TPMA_SESSION, the current unmarshaling code will place the input octet at the 0th octet of the TPMA_SESSION. Since the 0th octet is most significant octet, this has the effect of shifting all the session attribute bits left by 24 places.

As a consequence, someone implementing on a big endian machine should do one of two things:

a) allocate all structures as packed to a byte boundary (this may not be possible if the processor does not handle unaligned accesses); or

b) modify the code that manipulates bit fields that are not defined as being the alignment size of the system.

For many RISC processors, option #2 would be the only choice. This is may not be a terribly daunting task since only two attribute structures are not 32-bits (TPMA_SESSION and TPMA_LOCALITY).

5 Marshaling

5.1 Introduction

The marshaling and unmarshaling code and function prototypes are not listed, as the code is repetitive, long, and not very useful to read. Examples of a few unmarshaling routines are provided. Most of the others are similar.

Depending on the table header flags, a type will have an unmarshaling routine and a marshaling routine The table header flags that control the generation of the unmarshaling and marshaling code are delimited by angle brackets ("<>") in the table header. If no brackets are present, then both unmarshaling and marshaling code is generated (i.e., generation of both marshaling and unmarshaling code is the default).

5.2 Unmarshal and Marshal a Value

In TPM 2.0 Part 2, a TPMI_DI_OBJECT is defined by this table:

Table xxx — Definition of (TPM_HANDLE) TPMI_DH_OBJECT Type

Values	Comments
{TRANSIENT_FIRST:TRANSIENT_LAST}	allowed range for transient objects
{PERSISTENT_FIRST:PERSISTENT_LAST}	allowed range for persistent objects
+TPM_RH_NULL	the null handle
#TPM_RC_VALUE	

This generates the following unmarshaling code:

```
TPM_RC
TPMI_DH_OBJECT_Unmarshal(TPMI_DH_OBJECT *target, BYTE **buffer, INT32 *size,
                         BOOL flag)
{
    TPM_RC    result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE *)target, buffer, size);
    if(result != TPM_RC_SUCCESS)
        return result;
    if(*target == TPM_RH_NULL)
    {
        if(flag)
            return TPM_RC_SUCCESS;
        else
            return TPM_RC_VALUE;
    }
    if(((*target < TRANSIENT_FIRST) || (*target > TRANSIENT_LAST))
      &&((*target < PERSISTENT_FIRST) || (*target > PERSISTENT_LAST)))
            return TPM_RC_VALUE;
    return TPM_RC_SUCCESS;
}
```

and the following marshaling code:

NOTE The marshaling code does not do parameter checking, as the TPM is the source of the marshaling data.

```
UINT16
TPMI_DH_OBJECT_Marshal(TPMI_DH_OBJECT *source, BYTE **buffer, INT32 *size)
{
    return UINT32_Marshal((UINT32 *)source, buffer, size);
}
```

5.3 Unmarshal and Marshal a Union

In TPM 2.0 Part 2, a TPMU_PUBLIC_PARMS union is defined by:

Table xxx — Definition of TPMU_PUBLIC_PARMS Union <IN/OUT, S>

Parameter	Type	Selector	Description
keyedHash	TPMS_KEYEDHASH_PARMS	TPM_ALG_KEYEDHASH	sign | encrypt | neither
symDetail	TPMT_SYM_DEF_OBJECT	TPM_ALG_SYMCIPHER	a symmetric block cipher
rsaDetail	TPMS_RSA_PARMS	TPM_ALG_RSA	decrypt + sign
eccDetail	TPMS_ECC_PARMS	TPM_ALG_ECC	decrypt + sign
asymDetail	TPMS_ASYM_PARMS		common scheme structure for RSA and ECC keys
NOTE The Description column indicates which of TPMA_OBJECT.*decrypt* or TPMA_OBJECT.*sign* may be set. "+" indicates that both may be set but one shall be set. "|" indicates the optional settings.			

From this table, the following unmarshaling code is generated.

```
TPM_RC
TPMU_PUBLIC_PARMS_Unmarshal(TPMU_PUBLIC_PARMS *target, BYTE **buffer, INT32 *size,
                            UINT32 selector)
{
    switch(selector) {
#if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPMS_KEYEDHASH_PARMS_Unmarshal(
                        (TPMS_KEYEDHASH_PARMS *)&(target->keyedHash), buffer, size);
#endif
#if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPMT_SYM_DEF_OBJECT_Unmarshal(
                    (TPMT_SYM_DEF_OBJECT *)&(target->symDetail), buffer, size, FALSE);
#endif
#if ALG_RSA
        case TPM_ALG_RSA:
            return TPMS_RSA_PARMS_Unmarshal(
                                (TPMS_RSA_PARMS *)&(target->rsaDetail), buffer, size);
#endif
#if ALG_ECC
        case TPM_ALG_ECC:
            return TPMS_ECC_PARMS_Unmarshal(
                                (TPMS_ECC_PARMS *)&(target->eccDetail), buffer, size);
#endif
    }
    return TPM_RC_SELECTOR;
}
```

NOTE The `#if/#endif` directives are added whenever a value is dependent on an algorithm ID so that removing the algorithm definition will remove the related code.

The marshaling code for the union is:

```
UINT16
TPMU_PUBLIC_PARMS_Marshal(TPMU_PUBLIC_PARMS *source, BYTE **buffer, INT32 *size,
                          UINT32 selector)
{
    switch(selector) {
#if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
```

```
            return TPMS_KEYEDHASH_PARMS_Marshal(
                        (TPMS_KEYEDHASH_PARMS *)&(source->keyedHash), buffer, size);
#endif
#if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPMT_SYM_DEF_OBJECT_Marshal(
                        (TPMT_SYM_DEF_OBJECT *)&(source->symDetail), buffer, size);
#endif
#if ALG_RSA
        case TPM_ALG_RSA:
            return TPMS_RSA_PARMS_Marshal(
                        (TPMS_RSA_PARMS *)&(source->rsaDetail), buffer, size);
#endif
#if ALG_ECC
        case TPM_ALG_ECC:
            return TPMS_ECC_PARMS_Marshal(
                        (TPMS_ECC_PARMS *)&(source->eccDetail), buffer, size);
#endif
    }
    assert(1);
    return 0;
}
```

For the marshaling and unmarshaling code, a value in the structure containing the union provides the value used for *selector*. The example in the next section illustrates this.

5.4 Unmarshal and Marshal a Structure

In TPM 2.0 Part 2, the TPMT_PUBLIC structure is defined by:

Table xxx — Definition of TPMT_PUBLIC Structure

Parameter	Type	Description
type	TPMI_ALG_PUBLIC	“algorithm” associated with this object
nameAlg	+TPMI_ALG_HASH	algorithm used for computing the Name of the object NOTE The "+" indicates that the instance of a TPMT_PUBLIC may have a "+" to indicate that the nameAlg may be TPM_ALG_NULL.
objectAttributes	TPMA_OBJECT	attributes that, along with *type*, determine the manipulations of this object
authPolicy	TPM2B_DIGEST	optional policy for using this key The policy is computed using the *nameAlg* of the object. NOTE shall be the Empty Buffer if no authorization policy is present
[type]parameters	TPMU_PUBLIC_PARMS	the algorithm or structure details
[type]unique	TPMU_PUBLIC_ID	the unique identifier of the structure For an asymmetric key, this would be the public key.

This structure is tagged (the first value indicates the structure type), and that tag is used to determine how the parameters and unique fields are unmarshaled and marshaled. The use of the type for specifying the union selector is emphasized below.

The unmarshaling code for the structure in the table above is:

```
TPM_RC
TPMT_PUBLIC_Unmarshal(TPMT_PUBLIC *target, BYTE **buffer, INT32 *size, BOOL flag)
{
    TPM_RC    result;
    result = TPMI_ALG_PUBLIC_Unmarshal((TPMI_ALG_PUBLIC *)&(target->type),
                                       buffer, size);
```

```
    if(result != TPM_RC_SUCCESS)
        return result;
    result = TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH *)&(target->nameAlg),
                                     buffer, size, flag);
    if(result != TPM_RC_SUCCESS)
        return result;
    result = TPMA_OBJECT_Unmarshal((TPMA_OBJECT *)&(target->objectAttributes),
                                   buffer, size);
    if(result != TPM_RC_SUCCESS)
        return result;
    result = TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST *)&(target->authPolicy),
                                    buffer, size);
    if(result != TPM_RC_SUCCESS)
        return result;

    result = TPMU_PUBLIC_PARMS_Unmarshal((TPMU_PUBLIC_PARMS *)&(target->parameters),
                                         buffer, size, (UINT32)target->type);
    if(result != TPM_RC_SUCCESS)
        return result;

    result = TPMU_PUBLIC_ID_Unmarshal((TPMU_PUBLIC_ID *)&(target->unique),
                                      buffer, size, (UINT32)target->type);
    if(result != TPM_RC_SUCCESS)
        return result;

    return TPM_RC_SUCCESS;
}
```

The marshaling code for the TPMT_PUBLIC structure is:

```
UINT16
TPMT_PUBLIC_Marshal(TPMT_PUBLIC *source, BYTE **buffer, INT32 *size)
{
    UINT16    result = 0;
    result = (UINT16)(result + TPMI_ALG_PUBLIC_Marshal(
                                (TPMI_ALG_PUBLIC *)&(source->type), buffer, size));
    result = (UINT16)(result + TPMI_ALG_HASH_Marshal(
                               (TPMI_ALG_HASH *)&(source->nameAlg), buffer, size));
    result = (UINT16)(result + TPMA_OBJECT_Marshal(
                           (TPMA_OBJECT *)&(source->objectAttributes), buffer, size));

    result = (UINT16)(result + TPM2B_DIGEST_Marshal(
                             (TPM2B_DIGEST *)&(source->authPolicy), buffer, size));

    result = (UINT16)(result + TPMU_PUBLIC_PARMS_Marshal(
                           (TPMU_PUBLIC_PARMS *)&(source->parameters), buffer, size,
                                                  (UINT32)source->type));

    result = (UINT16)(result + TPMU_PUBLIC_ID_Marshal(
                               (TPMU_PUBLIC_ID *)&(source->unique), buffer, size,
                                                  (UINT32)source->type));

    return result;
}
```

5.5 Unmarshal and Marshal an Array

In TPM 2.0 Part 2, the TPML_DIGEST is defined by:

Table xxx — Definition of TPML_DIGEST Structure

Parameter	Type	Description
count {2:}	UINT32	number of digests in the list, minimum is two
digests[count]{:8}	TPM2B_DIGEST	a list of digests For TPM2_PolicyOR(), all digests will have been computed using the digest of the policy session. For TPM2_PCR_Read(), each digest will be the size of the digest for the bank containing the PCR.
#TPM_RC_SIZE		response code when count is not at least two or is greater than 8

The *digests* parameter is an array of up to *count* structures (TPM2B_DIGESTS). The auto-generated code to Unmarshal this structure is:

```
TPM_RC
TPML_DIGEST_Unmarshal(TPML_DIGEST *target, BYTE **buffer, INT32 *size)
{
    TPM_RC    result;
    result = UINT32_Unmarshal((UINT32 *)&(target->count), buffer, size);
    if(result != TPM_RC_SUCCESS)
        return result;

    if( (target->count < 2))     // This check is triggered by the {2:} notation
                                 // on 'count'
        return TPM_RC_SIZE;

    if((target->count) > 8)      // This check is triggered by the {:8} notation
                                 // on 'digests'.
        return TPM_RC_SIZE;

    result = TPM2B_DIGEST_Array_Unmarshal((TPM2B_DIGEST *)(target->digests),
                                          buffer, size, (INT32)(target->count));
    if(result != TPM_RC_SUCCESS)
        return result;

    return TPM_RC_SUCCESS;
}
```

The routine unmarshals a *count* value and passes that value to a routine that unmarshals an array of TPM2B_DIGEST values. The unmarshaling code for the array is:

```
TPM_RC
TPM2B_DIGEST_Array_Unmarshal(TPM2B_DIGEST *target, BYTE **buffer, INT32 *size,
                             INT32 count)
{
    TPM_RC    result;
    INT32 i;
    for(i = 0; i < count; i++) {
        result = TPM2B_DIGEST_Unmarshal(&target[i], buffer, size);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    return TPM_RC_SUCCESS;
}
```


Marshaling of the TPML_DIGEST uses a similar scheme with a structure specifying the number of elements in an array and a subsequent call to a routine to marshal an array of that type.

```
UINT16
TPML_DIGEST_Marshal(TPML_DIGEST *source, BYTE **buffer, INT32 *size)
{
    UINT16    result = 0;
    result = (UINT16)(result + UINT32_Marshal((UINT32 *)&(source->count), buffer,
                                                                          size));
    result = (UINT16)(result + TPM2B_DIGEST_Array_Marshal(
                                    (TPM2B_DIGEST *)(source->digests), buffer, size,
                                    (INT32)(source->count)));

    return result;
}
```

The marshaling code for the array is:

```
TPM_RC
TPM2B_DIGEST_Array_Unmarshal(TPM2B_DIGEST *target, BYTE **buffer, INT32 *size,
                             INT32 count)
{
    TPM_RC    result;
    INT32 i;
    for(i = 0; i < count; i++) {
        result = TPM2B_DIGEST_Unmarshal(&target[i], buffer, size);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    return TPM_RC_SUCCESS;
}
```

5.6 TPM2B Handling

A TPM2B structure is handled as a special case. The unmarshaling code is similar to what is shown in 5.5 but the unmarshaling/marshaling is to a union element. Each TPM2B is a union of two sized buffers, one of which is type specific (the ‘t’ element) and the other is a generic value (the ‘b’ element). This allows each of the TPM2B structures to have some inheritance property with all other TPM2B. The purpose is to allow functions that have parameters that can be any TPM2B structure while allowing other functions to be specific about the type of the TPM2B that is used. When the generic structure is allowed, the input parameter would use the ‘b’ element and when the type-specific structure is required, the ‘t’ element is used.

When marshaling a TPM2B where the second member is a BYTE array, the size parameter indicates the size of the array. The second member can also be a structure. In this case, the caller does not prefill the size member. The marshaling code must marshal the structure and then back fill the calculated size.

Table xxx — Definition of TPM2B_EVENT Structure

Parameter	Type	Description
size	UINT16	Size of the operand
buffer [size] {:1024}	BYTE	The operand

```
TPM_RC
TPM2B_EVENT_Unmarshal(TPM2B_EVENT *target, BYTE **buffer, INT32 *size)
{
    TPM_RC    result;
    result = UINT16_Unmarshal((UINT16 *)&(target->t.size), buffer, size);
    if(result != TPM_RC_SUCCESS)
```

```
        return result;
    // if size equal to 0, the rest of the structure is a zero buffer
    //  so stop processing
    if(target->t.size == 0)
        return TPM_RC_SUCCESS;
    if((target->t.size) > 1024)     // This check is triggered by the {:1024}
                                    // notation on 'buffer'
        return TPM_RC_SIZE;
    result = BYTE_Array_Unmarshal((BYTE *)(target->t.buffer), buffer, size,
                                  (INT32)(target->t.size));
    if(result != TPM_RC_SUCCESS)
        return result;
    return TPM_RC_SUCCESS;
}
```

using these structure definitions:

```
typedef union {
    struct {
        UINT16        size;
        BYTE          buffer[1024];
    }            t;
    TPM2B        b;
} TPM2B_EVENT;
```

6 TPM Reference Implementation Include Files

6.1 /tpm/include/platform_interface/pcrstruct.h

```
//
// This file defines the PCR and PCR_Attributes structures and
// related interface functions
//

#ifndef _PCRSTRUCT_H_
#define _PCRSTRUCT_H_

#include <public/BaseTypes.h>
#include <public/TpmAlgorithmDefines.h>
#include <public/TpmTypes.h>

// a single PCR
typedef struct
{
#if ALG_SHA1
    BYTE Sha1Pcr[SHA1_DIGEST_SIZE];
#endif
#if ALG_SHA256
    BYTE Sha256Pcr[SHA256_DIGEST_SIZE];
#endif
#if ALG_SHA384
    BYTE Sha384[SHA384_DIGEST_SIZE];
#endif
#if ALG_SHA512
    BYTE Sha512[SHA512_DIGEST_SIZE];
#endif
#if ALG_SM3_256
    BYTE Sm3_256[SM3_256_DIGEST_SIZE];
#endif
#if ALG_SHA3_256
    BYTE Sha3_256[SHA3_256_DIGEST_SIZE];
#endif
#if ALG_SHA3_384
    BYTE Sha3_384[SHA3_384_DIGEST_SIZE];
#endif
#if ALG_SHA3_512
    BYTE Sha3_512[SHA3_512_DIGEST_SIZE];
#endif
} PCR;

// see the comments below for supportsPolicyAuth to explain this
#define MAX_PCR_GROUP_BITS 3

typedef struct
{
    // SET if the PCR value should be saved in state save
    unsigned int stateSave : 1;

    // SET if the PCR is part of the "TCB group", causes the PCR counter not to
increment
    unsigned int doNotIncrementPcrCounter : 1;

    // PCRs may support policy or auth-value authorization.
    //
    // Such authorization values, if supported, are set by
    // TPM2_PCR_SetAuthPolicy and/or TPM2_PCR_SetAuthValue.
    //
    // PCRs that share the same policy/auth value are said to be in a "group".
    // PCRs that don't support authorization are said to be in group Zero.
```

```
    //
    // Group numbers are only used internally to indicate which PCRs share an
    // authorization value.  IOW the TPM client cannot refer to PCRs by group
    // number; the range of group numbers is implementation defined. zero
    // indicates the PCR doesn't support policy or auth verification.
    //
    // The size of this field must be large enough to support
    // NUM_POLICY_PCR_GROUP & NUM_AUTHVALUE_PCR_GROUP; the maximum number of groups
    // actually supported by this build of the core library.
    //
    // The number of bits allocated here does not control the number of groups,
    // but there is a static assert that the number of bits here is large
    // enough.
    unsigned int policyAuthGroup : MAX_PCR_GROUP_BITS;
    unsigned int authValuesGroup : MAX_PCR_GROUP_BITS;

    // these bitfields indicating the localities that can
    // reset or extend this PCR. A SET bit indicates the PCR can
    // be extended or reset from that locality.  The low-order bit in
    // each field is locality zero, and the high-order bit is locality 4.
    unsigned int resetLocality  : 5;
    unsigned int extendLocality : 5;
} PCR_Attributes;

// Get pointer to particular PCR from array if that PCR is allocated.
// otherwise returns NULL
BYTE* GetPcrPointerIfAllocated(PCR*       pPcrArray,
                               TPM_ALG_ID alg,       // IN: algorithm for bank
                               UINT32     pcrNumber  // IN: PCR number
);

// get a PCR pointer from the TPM's internal list, if it's allocated
// otherwise NULL
BYTE* GetPcrPointer(TPM_ALG_ID alg,       // IN: algorithm for bank
                    UINT32     pcrNumber  // IN: PCR number
);

#endif
```

6.2 /tpm/include/platform_interface/platform_to_tpm_interface.h

```
#include <platform_interface/prototypes/_TPM_Hash_Data_fp.h>
#include <platform_interface/prototypes/_TPM_Hash_End_fp.h>
#include <platform_interface/prototypes/_TPM_Hash_Start_fp.h>
#include <platform_interface/prototypes/_TPM_Init_fp.h>
#include <platform_interface/prototypes/ExecCommand_fp.h>
#include <platform_interface/prototypes/Manufacture_fp.h>
// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/prototypes/TpmFail_fp.h>
```

6.3 /tpm/include/platform_interface/tpm_to_platform_interface.h

```
// This file represents the functional interface that all platform libraries must
// provide because they are called by the Core TPM library.
#ifndef _TPM_TO_PLATFORM_INTERFACE_H_
#define _TPM_TO_PLATFORM_INTERFACE_H_

// need to read configuration for ACT_SUPPORT flag check below
#include <TpmConfiguration/TpmBuildSwitches.h>
#include <TpmConfiguration/TpmProfile.h>
#include <stddef.h>

//** From Cancel.c
```

```
//***_plat__IsCanceled()
// Check if the cancel flag is set
//  Return Type: int
//      TRUE(1)         if cancel flag is set
//      FALSE(0)        if cancel flag is not set
LIB_EXPORT int _plat__IsCanceled(void);

//***_plat__TimerRead()
// This function provides access to the tick timer of the platform. The TPM code
// uses this value to drive the TPM Clock.
//
// The tick timer is supposed to run when power is applied to the device. This timer
// should not be reset by time events including _TPM_Init. It should only be reset
// when TPM power is re-applied.
//
// If the TPM is run in a protected environment, that environment may provide the
// tick time to the TPM as long as the time provided by the environment is not
// allowed to go backwards. If the time provided by the system can go backwards
// during a power discontinuity, then the _plat__Signal_PowerOn should call
// _plat__TimerReset().
LIB_EXPORT uint64_t _plat__TimerRead(void);

//*** _plat__TimerWasReset()
// This function is used to interrogate the flag indicating if the tick timer has
// been reset.
//
// If the resetFlag parameter is SET, then the flag will be CLEAR before the
// function returns.
LIB_EXPORT int _plat__TimerWasReset(void);

//*** _plat__TimerWasStopped()
// This function is used to interrogate the flag indicating if the tick timer has
// been stopped. If so, this is typically a reason to roll the nonce.
//
// This function will CLEAR the s_timerStopped flag before returning. This provides
// functionality that is similar to status register that is cleared when read. This
// is the model used here because it is the one that has the most impact on the TPM
// code as the flag can only be accessed by one entity in the TPM. Any other
// implementation of the hardware can be made to look like a read-once register.
LIB_EXPORT int _plat__TimerWasStopped(void);

//***_plat__ClockRateAdjust()
// Adjust the clock rate
// the old function name is ClockAdjustRate, and took a value which was an absolute
// number of ticks.
//
// ClockRateAdjust uses predefined signal values and encapsulates the platform
// specifics regarding the number of ticks the underlying clock is running at.
//
// The adjustment must be one of these values. A COARSE adjustment is 1%, MEDIUM
// is 0.1%, and FINE is the smallest amount supported by the platform.  The
// total (cumulative) adjustment is limited to ~15% total.  Attempts to adjust
// the clock further are silently ignored as are any invalid values.  These
// values are defined here to insulate them from spec changes and to avoid
// needing visibility to the doc-generated structure headers.
typedef enum _plat__ClockAdjustStep
{
    PLAT_TPM_CLOCK_ADJUST_COARSE_SLOWER = -3,
    PLAT_TPM_CLOCK_ADJUST_MEDIUM_SLOWER = -2,
    PLAT_TPM_CLOCK_ADJUST_FINE_SLOWER   = -1,
    PLAT_TPM_CLOCK_ADJUST_FINE_FASTER   = 1,
    PLAT_TPM_CLOCK_ADJUST_MEDIUM_FASTER = 2,
    PLAT_TPM_CLOCK_ADJUST_COARSE_FASTER = 3
} _plat__ClockAdjustStep;
LIB_EXPORT void _plat__ClockRateAdjust(_plat__ClockAdjustStep adjustment);
```

```
//** From DebugHelpers.c

#if CERTIFYX509_DEBUG

//*** DebugFileInit()
// This function opens the file used to hold the debug data.
//  Return Type: int
//   0        success
//  != 0          error
int DebugFileInit(void);

//*** DebugDumpBuffer()
void DebugDumpBuffer(int size, unsigned char* buf, const char* identifier);
#endif  // CERTIFYX509_DEBUG

//** From Entropy.c

//*** _plat__GetEntropy()
// This function is used to get available hardware entropy. In a hardware
// implementation of this function, there would be no call to the system
// to get entropy.
//  Return Type: int32_t
//  < 0        hardware failure of the entropy generator, this is sticky
// >= 0        the returned amount of entropy (bytes)
//
LIB_EXPORT int32_t _plat__GetEntropy(unsigned char* entropy,  // output buffer
                                     uint32_t       amount    // amount requested
);

//** From LocalityPlat.c

//*** _plat__LocalityGet()
// Get the most recent command locality in locality value form.
// This is an integer value for locality and not a locality structure
// The locality can be 0-4 or 32-255. 5-31 is not allowed.
LIB_EXPORT unsigned char _plat__LocalityGet(void);

//*** _plat__NVEnable()
// Enable NV memory.
//
// This version just pulls in data from a file. In a real TPM, with NV on chip,
// this function would verify the integrity of the saved context. If the NV
// memory was not on chip but was in something like RPMB, the NV state would be
// read in, decrypted and integrity checked.
//
// The recovery from an integrity failure depends on where the error occurred. It
// it was in the state that is discarded by TPM Reset, then the error is
// recoverable if the TPM is reset. Otherwise, the TPM must go into failure mode.
//
//  Return Type: int
//      0           if success
//      >0          if recoverable error
//      <0          if unrecoverable error
LIB_EXPORT int _plat__NVEnable(
    void*  platParameter,  // platform specific parameter
    size_t paramSize       // size of parameter. If size == 0, then
                           // parameter is a sizeof(void*) scalar and should
                           // be cast to an integer (intptr_t), not dereferenced.
);

//*** _plat__GetNvReadyState()
// Check if NV is available
//  Return Type: int
//      0               NV is available
//      1               NV is not available due to write failure
//      2               NV is not available due to rate limit
```

```
#define NV_READY        0
#define NV_WRITEFAILURE 1
#define NV_RATE_LIMIT   2
LIB_EXPORT int _plat__GetNvReadyState(void);

//***_plat__NvMemoryRead()
// Function: Read a chunk of NV memory
//  Return Type: int
//      TRUE(1)         offset and size is within available NV size
//      FALSE(0)        otherwise; also trigger failure mode
LIB_EXPORT int _plat__NvMemoryRead(unsigned int startOffset,  // IN: read start
                                   unsigned int size,  // IN: size of bytes to read
                                   void*        data   // OUT: data buffer
);

//*** _plat__NvGetChangedStatus()
// This function checks to see if the NV is different from the test value. This is
// so that NV will not be written if it has not changed.
//  Return Type: int
//      NV_HAS_CHANGED(1)       the NV location is different from the test value
//      NV_IS_SAME(0)           the NV location is the same as the test value
//      NV_INVALID_LOCATION(-1) the NV location is invalid; also triggers failure mode
#define NV_HAS_CHANGED      (1)
#define NV_IS_SAME          (0)
#define NV_INVALID_LOCATION (-1)
LIB_EXPORT int _plat__NvGetChangedStatus(
    unsigned int startOffset,  // IN: read start
    unsigned int size,         // IN: size of bytes to read
    void*        data          // IN: data buffer
);

//***_plat__NvMemoryWrite()
// This function is used to update NV memory. The "write" is to a memory copy of
// NV. At the end of the current command, any changes are written to
// the actual NV memory.
// NOTE: A useful optimization would be for this code to compare the current
// contents of NV with the local copy and note the blocks that have changed. Then
// only write those blocks when _plat__NvCommit() is called.
//  Return Type: int
//      TRUE(1)         offset and size is within available NV size
//      FALSE(0)        otherwise; also trigger failure mode
LIB_EXPORT int _plat__NvMemoryWrite(unsigned int startOffset,  // IN: write start
                                    unsigned int size,  // IN: size of bytes to write
                                    void*        data   // OUT: data buffer
);

//***_plat__NvMemoryClear()
// Function is used to set a range of NV memory bytes to an implementation-dependent
// value. The value represents the erase state of the memory.
LIB_EXPORT int _plat__NvMemoryClear(unsigned int startOffset,  // IN: clear start
                                    unsigned int size  // IN: number of bytes to clear
);

//***_plat__NvMemoryMove()
// Function: Move a chunk of NV memory from source to destination
//      This function should ensure that if there overlap, the original data is
//      copied before it is written
LIB_EXPORT int _plat__NvMemoryMove(unsigned int sourceOffset,  // IN: source offset
                                   unsigned int destOffset,  // IN: destination offset
                                   unsigned int size  // IN: size of data being moved
);

//***_plat__NvCommit()
// This function writes the local copy of NV to NV for permanent store. It will write
// NV_MEMORY_SIZE bytes to NV. If a file is use, the entire file is written.
//  Return Type: int
```

```
//  0       NV write success
//  non-0   NV write fail
LIB_EXPORT int _plat__NvCommit(void);

//*** _plat__TearDown
// notify platform that TPM_TearDown was called so platform can cleanup or
// zeroize anything in the Platform.  This should zeroize NV as well.
LIB_EXPORT void _plat__TearDown();

//** From PlatformACT.c

#if ACT_SUPPORT
//*** _plat__ACT_GetImplemented()
// This function tests to see if an ACT is implemented. It is a belt and suspenders
// function because the TPM should not be calling to manipulate an ACT that is not
// implemented. However, this could help the simulator code which doesn't necessarily
// know if an ACT is implemented or not.
LIB_EXPORT int _plat__ACT_GetImplemented(uint32_t act);

//*** _plat__ACT_GetRemaining()
// This function returns the remaining time. If an update is pending, 'newValue' is
// returned. Otherwise, the current counter value is returned. Note that since the
// timers keep running, the returned value can get stale immediately. The actual count
// value will be no greater than the returned value.
LIB_EXPORT uint32_t _plat__ACT_GetRemaining(uint32_t act  //IN: the ACT selector
);

//*** _plat__ACT_GetSignaled()
LIB_EXPORT int _plat__ACT_GetSignaled(uint32_t act  //IN: number of ACT to check
);

//*** _plat__ACT_SetSignaled()
LIB_EXPORT void _plat__ACT_SetSignaled(uint32_t act, int on);

//*** _plat__ACT_UpdateCounter()
// This function is used to write the newValue for the counter. If an update is
// pending, then no update occurs and the function returns FALSE. If 'setSignaled'
// is TRUE, then the ACT signaled state is SET and if 'newValue' is 0, nothing
// is posted.
LIB_EXPORT int _plat__ACT_UpdateCounter(uint32_t act,      // IN: ACT to update
                                        uint32_t newValue  // IN: the value to post
);

//*** _plat__ACT_EnableTicks()
// This enables and disables the processing of the once-per-second ticks. This should
// be turned off ('enable' = FALSE) by _TPM_Init and turned on ('enable' = TRUE) by
// TPM2_Startup() after all the initializations have completed.
LIB_EXPORT void _plat__ACT_EnableTicks(int enable);

//*** _plat__ACT_Initialize()
// This function initializes the ACT hardware and data structures
LIB_EXPORT int _plat__ACT_Initialize(void);

#endif  // ACT_SUPPORT

//** From PowerPlat.c

//*** _plat__WasPowerLost()
// Test whether power was lost before a _TPM_Init.
//
// This function will clear the "hardware" indication of power loss before return.
// This means that there can only be one spot in the TPM code where this value
// gets read. This method is used here as it is the most difficult to manage in the
// TPM code and, if the hardware actually works this way, it is hard to make it
// look like anything else. So, the burden is placed on the TPM code rather than the
// platform code
```

```
//  Return Type: int
//      TRUE(1)         power was lost
//      FALSE(0)        power was not lost
LIB_EXPORT int _plat__WasPowerLost(void);

//** From PPPlat.c

//*** _plat__PhysicalPresenceAsserted()
// Check if physical presence is signaled
//  Return Type: int
//      TRUE(1)         if physical presence is signaled
//      FALSE(0)        if physical presence is not signaled
LIB_EXPORT int _plat__PhysicalPresenceAsserted(void);

//*** _plat__Fail()
// This is the platform depended failure exit for the TPM.
LIB_EXPORT NORETURN void _plat__Fail(void);

//** From Unique.c

#if VENDOR_PERMANENT_AUTH_ENABLED == YES
//** _plat__GetUnique()
// This function is used to access the platform-specific unique values.
// This function places the unique value in the provided buffer ('b')
// and returns the number of bytes transferred. The function will not
// copy more data than 'bSize'.
// zero indicates value does not exist or an error occurred.
//
// 'which' indicates the unique value to return:
// 0 = RESERVED, do not use
// 1 = the VENDOR_PERMANENT_AUTH_HANDLE authorization value for this device
LIB_EXPORT uint32_t _plat__GetUnique(uint32_t       which,
                                     uint32_t       bSize,  // size of the buffer
                                     unsigned char* b       // output buffer
);
#endif

//** _plat__GetPlatformManufactureData
// This function allows the platform to provide a small amount of data to be
// stored as part of the TPM's PERSISTENT_DATA structure during manufacture.  Of
// course the platform can store data separately as well, but this allows a
// simple platform implementation to store a few bytes of data without
// implementing a multi-layer storage system.  This function is called on
// manufacture and CLEAR.  The buffer will contain the last value provided
// to the Core library.
LIB_EXPORT void _plat__GetPlatformManufactureData(uint8_t* pPlatformPersistentData,
                                                  uint32_t bufferSize);

// return the 4 character Manufacturer Capability code.  This
// should come from the platform library since that is provided by the manufacturer
LIB_EXPORT uint32_t _plat__GetManufacturerCapabilityCode();

// return the 4 character VendorStrings for Capabilities.
// Index is ONE-BASED, and may be in the range [1,4] inclusive.
// Any other index returns all zeros. The return value will be interpreted
// as an array of 4 ASCII characters (with no null terminator)
LIB_EXPORT uint32_t _plat__GetVendorCapabilityCode(int index);

// return the most-significant 32-bits of the TPM Firmware Version reported by
// getCapability.
LIB_EXPORT uint32_t _plat__GetTpmFirmwareVersionHigh();

// return the least-significant 32-bits of the TPM Firmware Version reported by
// getCapability.
LIB_EXPORT uint32_t _plat__GetTpmFirmwareVersionLow();
```

```
// return the TPM Firmware's current SVN.
LIB_EXPORT uint16_t _plat__GetTpmFirmwareSvn(void);

// return the maximum value that the TPM Firmware SVN may take.
LIB_EXPORT uint16_t _plat__GetTpmFirmwareMaxSvn(void);

#if SVN_LIMITED_SUPPORT
//***_plat__GetTpmFirmwareSvnSecret()
// Function: Obtain a Firmware SVN Secret bound to the given SVN. Fails if the
// given SVN is greater than the firmware's current SVN.
// size must equal PRIMARY_SEED_SIZE.
// Return Type: int
//  0           success
//  != 0        error
LIB_EXPORT int _plat__GetTpmFirmwareSvnSecret(
    uint16_t  svn,              // IN: specified SVN
    uint16_t  secret_buf_size,  // IN: size of secret buffer
    uint8_t*  secret_buf,       // OUT: secret buffer
    uint16_t* secret_size       // OUT: secret buffer
);
#endif  // SVN_LIMITED_SUPPORT

#if FW_LIMITED_SUPPORT
//***_plat__GetTpmFirmwareSecret()
// Function: Obtain a Firmware Secret bound to the current firmware image.
// Return Type: int
//  0           success
//  != 0        error
LIB_EXPORT int _plat__GetTpmFirmwareSecret(
    uint16_t  secret_buf_size,  // IN: size of secret buffer
    uint8_t*  secret_buf,       // OUT: secret buffer
    uint16_t* secret_size       // OUT: secret buffer
);
#endif  // FW_LIMITED_SUPPORT

// return the TPM Type returned by TPM_PT_VENDOR_TPM_TYPE
LIB_EXPORT uint32_t _plat__GetTpmType();

// platform PCR initialization functions
#include <platform_interface/prototypes/platform_pcr_fp.h>

#endif  // _TPM_TO_PLATFORM_INTERFACE_H_
```

6.4 /tpm/include/platform_interface/prototypes/ExecCommand_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _EXEC_COMMAND_FP_H_
#define _EXEC_COMMAND_FP_H_

//** ExecuteCommand()
//
// The function performs the following steps.
//
//  a)  Parses the command header from input buffer.
//  b)  Calls ParseHandleBuffer() to parse the handle area of the command.
//  c)  Validates that each of the handles references a loaded entity.
//  d)  Calls ParseSessionBuffer () to:
//      1)  unmarshal and parse the session area;
//      2)  check the authorizations; and
//      3)  when necessary, decrypt a parameter.
//  e)  Calls CommandDispatcher() to:
```

```
//      1)  unmarshal the command parameters from the command buffer;
//      2)  call the routine that performs the command actions; and
//      3)  marshal the responses into the response buffer.
//  f)  If any error occurs in any of the steps above create the error response
//      and return.
//  g)  Calls BuildResponseSession() to:
//      1)  when necessary, encrypt a parameter
//      2)  build the response authorization sessions
//      3)  update the audit sessions and nonces
//  h)  Calls BuildResponseHeader() to complete the construction of the response.
//
// 'responseSize' is set by the caller to the maximum number of bytes available in
// the output buffer. ExecuteCommand will adjust the value and return the number
// of bytes placed in the buffer.
//
// 'response' is also set by the caller to indicate the buffer into which
//  ExecuteCommand is to place the response.
//
//  'request' and 'response' may point to the same buffer
//
// Note: As of February, 2016, the failure processing has been moved to the
// platform-specific code. When the TPM code encounters an unrecoverable failure, it
// will SET g_inFailureMode and call _plat__Fail(). That function should not return
// but may call ExecuteCommand().
//
LIB_EXPORT void ExecuteCommand(
    uint32_t        requestSize,    // IN: command buffer size
    unsigned char*  request,        // IN: command buffer
    uint32_t*       responseSize,   // IN/OUT: response buffer size
    unsigned char** response        // IN/OUT: response buffer
);

#endif  // _EXEC_COMMAND_FP_H_
```

6.5 /tpm/include/platform_interface/prototypes/Manufacture_fp.h

```
#ifndef _MANUFACTURE_FP_H_
#define _MANUFACTURE_FP_H_

//*** TPM_Manufacture()
// This function initializes the TPM values in preparation for the TPM's first
// use. This function will fail if previously called. The TPM can be re-manufactured
// by calling TPM_Teardown() first and then calling this function again.
// NV must be enabled first (typically with NvPowerOn() via _TPM_Init)
//
// return type: int
//      -2          NV System not available
//      -1          FAILURE - System is incorrectly compiled.
//      0           success
//      1           manufacturing process previously performed
// returns
#define MANUF_NV_NOT_READY   (-2)
#define MANUF_INVALID_CONFIG (-1)
#define MANUF_OK              0
#define MANUF_ALREADY_DONE    1
// params
#define MANUF_FIRST_TIME    1
#define MANUF_REMANUFACTURE 0
LIB_EXPORT int TPM_Manufacture(
    int firstTime  // IN: indicates if this is the first call from
                   //     main()
);

//*** TPM_TearDown()
// This function prepares the TPM for re-manufacture. It should not be implemented
```

```
// in anything other than a simulated TPM.
//
// In this implementation, all that is needs is to stop the cryptographic units
// and set a flag to indicate that the TPM can be re-manufactured. This should
// be all that is necessary to start the manufacturing process again.
//  Return Type: int
//      0        success
//      1        TPM not previously manufactured
#define TEARDOWN_OK          0
#define TEARDOWN_NOTHINGDONE 1
LIB_EXPORT int TPM_TearDown(void);

//*** TpmEndSimulation()
// This function is called at the end of the simulation run. It is used to provoke
// printing of any statistics that might be needed.
LIB_EXPORT void TpmEndSimulation(void);

#endif  // _MANUFACTURE_FP_H_
```

6.6 /tpm/include/platform_interface/prototypes/platform_pcr_fp.h

```
// platform PCR functions called by the TPM library

#ifndef _PLATFORM_PCR_FP_H_
#define _PLATFORM_PCR_FP_H_

#include <public/BaseTypes.h>
#include <public/TpmTypes.h>
#include <platform_interface/pcrstruct.h>

// return the number of PCRs the platform recognizes for
GetPcrInitializationAttributes.
// PCRs are numbered starting at zero.
// Note: The TPM Library will enter failure mode if this number doesn't match
// IMPLEMENTATION_PCR.
UINT32 _platPcr__NumberOfPcrs(void);

// return the initialization attributes of a given PCR.
// pcrNumber expected to be in [0, _platPcr__NumberOfPcrs)
// returns the attributes for PCR[0] if the requested pcrNumber is out of range.
// Note this returns a structure by-value, which is fast because the structure is
// a bitfield.
PCR_Attributes _platPcr__GetPcrInitializationAttributes(UINT32 pcrNumber);

// Fill a given buffer with the PCR initialization value for a particular PCR and hash
// combination, and return its length.  If the platform doesn't have a value, then
// the result size is expected to be zero, and the rfunction will return TPM_RC_PCR.
// If a valid is not available, then the core TPM library will ignore the value and
// treat it as non-existant and provide a default.
// If the buffer is not large enough for a pcr consistent with pcrAlg, then the
// platform will return TPM_RC_FAILURE.
TPM_RC _platPcr__GetInitialValueForPcr(
    UINT32     pcrNumber,        // IN: PCR to be initialized
    TPM_ALG_ID pcrAlg,           // IN: Algorithm of the PCR Bank being initialized
    BYTE       startupLocality,  // IN: locality where startup is being called from
    BYTE*      pcrBuffer,        // OUT: buffer to put PCR initialization value into
    uint16_t   bufferSize,       // IN: maximum size of value buffer can hold
    uint16_t*  pcrLength);  // OUT: size of initialization value returned in pcrBuffer

// should the given PCR algorithm default to active in a new TPM?
BOOL _platPcr_IsPcrBankDefaultActive(TPM_ALG_ID pcrAlg);

#endif  // _PLATFORM_PCR_FP_H_
```

6.7 /tpm/include/platform_interface/prototypes/_TPM_Hash_Data_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef __TPM_HASH_DATA_FP_H_
#define __TPM_HASH_DATA_FP_H_

// This function is called to process a _TPM_Hash_Data indication.
LIB_EXPORT void _TPM_Hash_Data(uint32_t dataSize,   // IN: size of data to be extend
                               unsigned char* data  // IN: data buffer
);

#endif  // __TPM_HASH_DATA_FP_H_
```

6.8 /tpm/include/platform_interface/prototypes/_TPM_Hash_End_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef __TPM_HASH_END_FP_H_
#define __TPM_HASH_END_FP_H_

// This function is called to process a _TPM_Hash_End indication.
LIB_EXPORT void _TPM_Hash_End(void);

#endif  // __TPM_HASH_END_FP_H_
```

6.9 /tpm/include/platform_interface/prototypes/_TPM_Hash_Start_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef __TPM_HASH_START_FP_H_
#define __TPM_HASH_START_FP_H_

// This function is called to process a _TPM_Hash_Start indication.
LIB_EXPORT void _TPM_Hash_Start(void);

#endif  // __TPM_HASH_START_FP_H_
```

6.10 /tpm/include/platform_interface/prototypes/_TPM_Init_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef __TPM_INIT_FP_H_
#define __TPM_INIT_FP_H_

// This function is used to process a _TPM_Init indication.
LIB_EXPORT void _TPM_Init(void);

#endif  // __TPM_INIT_FP_H_
```

6.11 /tpm/include/private/CommandAttributeData.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT
// clang-format off

// This file should only be included by CommandCodeAttibutes.c
#ifdef _COMMAND_CODE_ATTRIBUTES_

#include "CommandAttributes.h"

#if COMPRESSED_LISTS
#   define      PAD_LIST    0
#else
#   define      PAD_LIST    1
#endif

// This is the command code attribute array for GetCapability.
// Both this array and s_commandAttributes provides command code attributes,
// but tuned for different purpose
const TPMA_CC    s_ccAttr [] = {
#if (PAD_LIST || CC_NV_UndefineSpaceSpecial)
        TPMA_CC_INITIALIZER(0x011F, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_EvictControl)
        TPMA_CC_INITIALIZER(0x0120, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_HierarchyControl)
        TPMA_CC_INITIALIZER(0x0121, 0, 1, 1, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_UndefineSpace)
        TPMA_CC_INITIALIZER(0x0122, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST)
        TPMA_CC_INITIALIZER(0x0123, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ChangeEPS)
        TPMA_CC_INITIALIZER(0x0124, 0, 1, 1, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ChangePPS)
        TPMA_CC_INITIALIZER(0x0125, 0, 1, 1, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Clear)
        TPMA_CC_INITIALIZER(0x0126, 0, 1, 1, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ClearControl)
        TPMA_CC_INITIALIZER(0x0127, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ClockSet)
        TPMA_CC_INITIALIZER(0x0128, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_HierarchyChangeAuth)
        TPMA_CC_INITIALIZER(0x0129, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_DefineSpace)
        TPMA_CC_INITIALIZER(0x012A, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PCR_Allocate)
        TPMA_CC_INITIALIZER(0x012B, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PCR_SetAuthPolicy)
        TPMA_CC_INITIALIZER(0x012C, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PP_Commands)
        TPMA_CC_INITIALIZER(0x012D, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_SetPrimaryPolicy)
```

```
        TPMA_CC_INITIALIZER(0x012E, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_FieldUpgradeStart)
        TPMA_CC_INITIALIZER(0x012F, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ClockRateAdjust)
        TPMA_CC_INITIALIZER(0x0130, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_CreatePrimary)
        TPMA_CC_INITIALIZER(0x0131, 0, 0, 0, 0, 1, 1, 0, 0),
#endif
#if (PAD_LIST || CC_NV_GlobalWriteLock)
        TPMA_CC_INITIALIZER(0x0132, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_GetCommandAuditDigest)
        TPMA_CC_INITIALIZER(0x0133, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_Increment)
        TPMA_CC_INITIALIZER(0x0134, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_SetBits)
        TPMA_CC_INITIALIZER(0x0135, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_Extend)
        TPMA_CC_INITIALIZER(0x0136, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_Write)
        TPMA_CC_INITIALIZER(0x0137, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_WriteLock)
        TPMA_CC_INITIALIZER(0x0138, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_DictionaryAttackLockReset)
        TPMA_CC_INITIALIZER(0x0139, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_DictionaryAttackParameters)
        TPMA_CC_INITIALIZER(0x013A, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_ChangeAuth)
        TPMA_CC_INITIALIZER(0x013B, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PCR_Event)
        TPMA_CC_INITIALIZER(0x013C, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PCR_Reset)
        TPMA_CC_INITIALIZER(0x013D, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_SequenceComplete)
        TPMA_CC_INITIALIZER(0x013E, 0, 0, 0, 1, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_SetAlgorithmSet)
        TPMA_CC_INITIALIZER(0x013F, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_SetCommandCodeAuditStatus)
        TPMA_CC_INITIALIZER(0x0140, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_FieldUpgradeData)
        TPMA_CC_INITIALIZER(0x0141, 0, 1, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_IncrementalSelfTest)
        TPMA_CC_INITIALIZER(0x0142, 0, 1, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_SelfTest)
        TPMA_CC_INITIALIZER(0x0143, 0, 1, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Startup)
```

```
        TPMA_CC_INITIALIZER(0x0144, 0, 1, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Shutdown)
        TPMA_CC_INITIALIZER(0x0145, 0, 1, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_StirRandom)
        TPMA_CC_INITIALIZER(0x0146, 0, 1, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ActivateCredential)
        TPMA_CC_INITIALIZER(0x0147, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Certify)
        TPMA_CC_INITIALIZER(0x0148, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyNV)
        TPMA_CC_INITIALIZER(0x0149, 0, 0, 0, 0, 3, 0, 0, 0),
#endif
#if (PAD_LIST || CC_CertifyCreation)
        TPMA_CC_INITIALIZER(0x014A, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Duplicate)
        TPMA_CC_INITIALIZER(0x014B, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_GetTime)
        TPMA_CC_INITIALIZER(0x014C, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_GetSessionAuditDigest)
        TPMA_CC_INITIALIZER(0x014D, 0, 0, 0, 0, 3, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_Read)
        TPMA_CC_INITIALIZER(0x014E, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_ReadLock)
        TPMA_CC_INITIALIZER(0x014F, 0, 1, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ObjectChangeAuth)
        TPMA_CC_INITIALIZER(0x0150, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicySecret)
        TPMA_CC_INITIALIZER(0x0151, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Rewrap)
        TPMA_CC_INITIALIZER(0x0152, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Create)
        TPMA_CC_INITIALIZER(0x0153, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ECDH_ZGen)
        TPMA_CC_INITIALIZER(0x0154, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || (CC_HMAC || CC_MAC))
        TPMA_CC_INITIALIZER(0x0155, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Import)
        TPMA_CC_INITIALIZER(0x0156, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Load)
        TPMA_CC_INITIALIZER(0x0157, 0, 0, 0, 0, 1, 1, 0, 0),
#endif
#if (PAD_LIST || CC_Quote)
        TPMA_CC_INITIALIZER(0x0158, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_RSA_Decrypt)
        TPMA_CC_INITIALIZER(0x0159, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST)
```

```
        TPMA_CC_INITIALIZER(0x015A, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || (CC_HMAC_Start || CC_MAC_Start))
        TPMA_CC_INITIALIZER(0x015B, 0, 0, 0, 0, 1, 1, 0, 0),
#endif
#if (PAD_LIST || CC_SequenceUpdate)
        TPMA_CC_INITIALIZER(0x015C, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Sign)
        TPMA_CC_INITIALIZER(0x015D, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Unseal)
        TPMA_CC_INITIALIZER(0x015E, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST)
        TPMA_CC_INITIALIZER(0x015F, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicySigned)
        TPMA_CC_INITIALIZER(0x0160, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ContextLoad)
        TPMA_CC_INITIALIZER(0x0161, 0, 0, 0, 0, 0, 1, 0, 0),
#endif
#if (PAD_LIST || CC_ContextSave)
        TPMA_CC_INITIALIZER(0x0162, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ECDH_KeyGen)
        TPMA_CC_INITIALIZER(0x0163, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_EncryptDecrypt)
        TPMA_CC_INITIALIZER(0x0164, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_FlushContext)
        TPMA_CC_INITIALIZER(0x0165, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST)
        TPMA_CC_INITIALIZER(0x0166, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_LoadExternal)
        TPMA_CC_INITIALIZER(0x0167, 0, 0, 0, 0, 0, 1, 0, 0),
#endif
#if (PAD_LIST || CC_MakeCredential)
        TPMA_CC_INITIALIZER(0x0168, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_ReadPublic)
        TPMA_CC_INITIALIZER(0x0169, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyAuthorize)
        TPMA_CC_INITIALIZER(0x016A, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyAuthValue)
        TPMA_CC_INITIALIZER(0x016B, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyCommandCode)
        TPMA_CC_INITIALIZER(0x016C, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyCounterTimer)
        TPMA_CC_INITIALIZER(0x016D, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyCpHash)
        TPMA_CC_INITIALIZER(0x016E, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyLocality)
        TPMA_CC_INITIALIZER(0x016F, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyNameHash)
```

```
        TPMA_CC_INITIALIZER(0x0170, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyOR)
        TPMA_CC_INITIALIZER(0x0171, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyTicket)
        TPMA_CC_INITIALIZER(0x0172, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ReadPublic)
        TPMA_CC_INITIALIZER(0x0173, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_RSA_Encrypt)
        TPMA_CC_INITIALIZER(0x0174, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST)
        TPMA_CC_INITIALIZER(0x0175, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_StartAuthSession)
        TPMA_CC_INITIALIZER(0x0176, 0, 0, 0, 0, 2, 1, 0, 0),
#endif
#if (PAD_LIST || CC_VerifySignature)
        TPMA_CC_INITIALIZER(0x0177, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ECC_Parameters)
        TPMA_CC_INITIALIZER(0x0178, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_FirmwareRead)
        TPMA_CC_INITIALIZER(0x0179, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_GetCapability)
        TPMA_CC_INITIALIZER(0x017A, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_GetRandom)
        TPMA_CC_INITIALIZER(0x017B, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_GetTestResult)
        TPMA_CC_INITIALIZER(0x017C, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Hash)
        TPMA_CC_INITIALIZER(0x017D, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PCR_Read)
        TPMA_CC_INITIALIZER(0x017E, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyPCR)
        TPMA_CC_INITIALIZER(0x017F, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyRestart)
        TPMA_CC_INITIALIZER(0x0180, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ReadClock)
        TPMA_CC_INITIALIZER(0x0181, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PCR_Extend)
        TPMA_CC_INITIALIZER(0x0182, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PCR_SetAuthValue)
        TPMA_CC_INITIALIZER(0x0183, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_Certify)
        TPMA_CC_INITIALIZER(0x0184, 0, 0, 0, 0, 3, 0, 0, 0),
#endif
#if (PAD_LIST || CC_EventSequenceComplete)
        TPMA_CC_INITIALIZER(0x0185, 0, 1, 0, 1, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_HashSequenceStart)
```

```
        TPMA_CC_INITIALIZER(0x0186, 0, 0, 0, 0, 0, 1, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyPhysicalPresence)
        TPMA_CC_INITIALIZER(0x0187, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyDuplicationSelect)
        TPMA_CC_INITIALIZER(0x0188, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyGetDigest)
        TPMA_CC_INITIALIZER(0x0189, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_TestParms)
        TPMA_CC_INITIALIZER(0x018A, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Commit)
        TPMA_CC_INITIALIZER(0x018B, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyPassword)
        TPMA_CC_INITIALIZER(0x018C, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ZGen_2Phase)
        TPMA_CC_INITIALIZER(0x018D, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_EC_Ephemeral)
        TPMA_CC_INITIALIZER(0x018E, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyNvWritten)
        TPMA_CC_INITIALIZER(0x018F, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyTemplate)
        TPMA_CC_INITIALIZER(0x0190, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_CreateLoaded)
        TPMA_CC_INITIALIZER(0x0191, 0, 0, 0, 0, 1, 1, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyAuthorizeNV)
        TPMA_CC_INITIALIZER(0x0192, 0, 0, 0, 0, 3, 0, 0, 0),
#endif
#if (PAD_LIST || CC_EncryptDecrypt2)
        TPMA_CC_INITIALIZER(0x0193, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_AC_GetCapability)
        TPMA_CC_INITIALIZER(0x0194, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_AC_Send)
        TPMA_CC_INITIALIZER(0x0195, 0, 0, 0, 0, 3, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Policy_AC_SendSelect)
        TPMA_CC_INITIALIZER(0x0196, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_CertifyX509)
        TPMA_CC_INITIALIZER(0x0197, 0, 0, 0, 0, 2, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ACT_SetTimeout)
        TPMA_CC_INITIALIZER(0x0198, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ECC_Encrypt)
        TPMA_CC_INITIALIZER(0x0199, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_ECC_Decrypt)
        TPMA_CC_INITIALIZER(0x019A, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyCapability)
        TPMA_CC_INITIALIZER(0x019B, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_PolicyParameters)
```

```
        TPMA_CC_INITIALIZER(0x019C, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_DefineSpace2)
        TPMA_CC_INITIALIZER(0x019D, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_NV_ReadPublic2)
        TPMA_CC_INITIALIZER(0x019E, 0, 0, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_SetCapability)
        TPMA_CC_INITIALIZER(0x019F, 0, 1, 0, 0, 1, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Vendor_TCG_Test)
        TPMA_CC_INITIALIZER(0x0000, 0, 0, 0, 0, 0, 0, 1, 0),
#endif
        TPMA_ZERO_INITIALIZER()
};


// This is the command code attribute structure.
const COMMAND_ATTRIBUTES    s_commandAttributes [] = {
#if (PAD_LIST || CC_NV_UndefineSpaceSpecial)
        (COMMAND_ATTRIBUTES)(CC_NV_UndefineSpaceSpecial * // 0x011F
            (IS_IMPLEMENTED+HANDLE_1_ADMIN+HANDLE_2_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_EvictControl)
        (COMMAND_ATTRIBUTES)(CC_EvictControl * // 0x0120
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_HierarchyControl)
        (COMMAND_ATTRIBUTES)(CC_HierarchyControl * // 0x0121
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_NV_UndefineSpace)
        (COMMAND_ATTRIBUTES)(CC_NV_UndefineSpace * // 0x0122
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST)
        (COMMAND_ATTRIBUTES)(0), // 0x0123
#endif
#if (PAD_LIST || CC_ChangeEPS)
        (COMMAND_ATTRIBUTES)(CC_ChangeEPS * // 0x0124
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_ChangePPS)
        (COMMAND_ATTRIBUTES)(CC_ChangePPS * // 0x0125
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_Clear)
        (COMMAND_ATTRIBUTES)(CC_Clear * // 0x0126
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_ClearControl)
        (COMMAND_ATTRIBUTES)(CC_ClearControl * // 0x0127
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_ClockSet)
        (COMMAND_ATTRIBUTES)(CC_ClockSet * // 0x0128
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_HierarchyChangeAuth)
        (COMMAND_ATTRIBUTES)(CC_HierarchyChangeAuth * // 0x0129
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_NV_DefineSpace)
        (COMMAND_ATTRIBUTES)(CC_NV_DefineSpace * // 0x012A
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
```

```
#endif
#if (PAD_LIST || CC_PCR_Allocate)
        (COMMAND_ATTRIBUTES)(CC_PCR_Allocate * // 0x012B
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_PCR_SetAuthPolicy)
        (COMMAND_ATTRIBUTES)(CC_PCR_SetAuthPolicy * // 0x012C
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_PP_Commands)
        (COMMAND_ATTRIBUTES)(CC_PP_Commands * // 0x012D
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_REQUIRED)),
#endif
#if (PAD_LIST || CC_SetPrimaryPolicy)
        (COMMAND_ATTRIBUTES)(CC_SetPrimaryPolicy * // 0x012E
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_FieldUpgradeStart)
        (COMMAND_ATTRIBUTES)(CC_FieldUpgradeStart * // 0x012F
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_ClockRateAdjust)
        (COMMAND_ATTRIBUTES)(CC_ClockRateAdjust * // 0x0130
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_CreatePrimary)
        (COMMAND_ATTRIBUTES)(CC_CreatePrimary * // 0x0131
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND+ENCRYPT_2+R_HANDLE)),
#endif
#if (PAD_LIST || CC_NV_GlobalWriteLock)
        (COMMAND_ATTRIBUTES)(CC_NV_GlobalWriteLock * // 0x0132
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_GetCommandAuditDigest)
        (COMMAND_ATTRIBUTES)(CC_GetCommandAuditDigest * // 0x0133
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_NV_Increment)
        (COMMAND_ATTRIBUTES)(CC_NV_Increment * // 0x0134
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_NV_SetBits)
        (COMMAND_ATTRIBUTES)(CC_NV_SetBits * // 0x0135
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_NV_Extend)
        (COMMAND_ATTRIBUTES)(CC_NV_Extend * // 0x0136
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_NV_Write)
        (COMMAND_ATTRIBUTES)(CC_NV_Write * // 0x0137
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_NV_WriteLock)
        (COMMAND_ATTRIBUTES)(CC_NV_WriteLock * // 0x0138
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_DictionaryAttackLockReset)
        (COMMAND_ATTRIBUTES)(CC_DictionaryAttackLockReset * // 0x0139
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_DictionaryAttackParameters)
        (COMMAND_ATTRIBUTES)(CC_DictionaryAttackParameters * // 0x013A
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_NV_ChangeAuth)
```

```
        (COMMAND_ATTRIBUTES)(CC_NV_ChangeAuth * // 0x013B
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN)),
#endif
#if (PAD_LIST || CC_PCR_Event)
        (COMMAND_ATTRIBUTES)(CC_PCR_Event * // 0x013C
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_PCR_Reset)
        (COMMAND_ATTRIBUTES)(CC_PCR_Reset * // 0x013D
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_SequenceComplete)
        (COMMAND_ATTRIBUTES)(CC_SequenceComplete * // 0x013E
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_SetAlgorithmSet)
        (COMMAND_ATTRIBUTES)(CC_SetAlgorithmSet * // 0x013F
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_SetCommandCodeAuditStatus)
        (COMMAND_ATTRIBUTES)(CC_SetCommandCodeAuditStatus * // 0x0140
            (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_FieldUpgradeData)
        (COMMAND_ATTRIBUTES)(CC_FieldUpgradeData * // 0x0141
            (IS_IMPLEMENTED+DECRYPT_2)),
#endif
#if (PAD_LIST || CC_IncrementalSelfTest)
        (COMMAND_ATTRIBUTES)(CC_IncrementalSelfTest * // 0x0142
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_SelfTest)
        (COMMAND_ATTRIBUTES)(CC_SelfTest * // 0x0143
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_Startup)
        (COMMAND_ATTRIBUTES)(CC_Startup * // 0x0144
            (IS_IMPLEMENTED+NO_SESSIONS)),
#endif
#if (PAD_LIST || CC_Shutdown)
        (COMMAND_ATTRIBUTES)(CC_Shutdown * // 0x0145
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_StirRandom)
        (COMMAND_ATTRIBUTES)(CC_StirRandom * // 0x0146
            (IS_IMPLEMENTED+DECRYPT_2)),
#endif
#if (PAD_LIST || CC_ActivateCredential)
        (COMMAND_ATTRIBUTES)(CC_ActivateCredential * // 0x0147
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Certify)
        (COMMAND_ATTRIBUTES)(CC_Certify * // 0x0148
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicyNV)
        (COMMAND_ATTRIBUTES)(CC_PolicyNV * // 0x0149
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_CertifyCreation)
        (COMMAND_ATTRIBUTES)(CC_CertifyCreation * // 0x014A
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Duplicate)
        (COMMAND_ATTRIBUTES)(CC_Duplicate * // 0x014B
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_DUP+ENCRYPT_2)),
```

```
#endif
#if (PAD_LIST || CC_GetTime)
        (COMMAND_ATTRIBUTES)(CC_GetTime * // 0x014C
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_GetSessionAuditDigest)
        (COMMAND_ATTRIBUTES)(CC_GetSessionAuditDigest * // 0x014D
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_NV_Read)
        (COMMAND_ATTRIBUTES)(CC_NV_Read * // 0x014E
            (IS_IMPLEMENTED+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_NV_ReadLock)
        (COMMAND_ATTRIBUTES)(CC_NV_ReadLock * // 0x014F
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_ObjectChangeAuth)
        (COMMAND_ATTRIBUTES)(CC_ObjectChangeAuth * // 0x0150
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicySecret)
        (COMMAND_ATTRIBUTES)(CC_PolicySecret * // 0x0151
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ALLOW_TRIAL+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Rewrap)
        (COMMAND_ATTRIBUTES)(CC_Rewrap * // 0x0152
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Create)
        (COMMAND_ATTRIBUTES)(CC_Create * // 0x0153
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_ECDH_ZGen)
        (COMMAND_ATTRIBUTES)(CC_ECDH_ZGen * // 0x0154
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || (CC_HMAC || CC_MAC))
        (COMMAND_ATTRIBUTES)((CC_HMAC || CC_MAC) * // 0x0155
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Import)
        (COMMAND_ATTRIBUTES)(CC_Import * // 0x0156
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Load)
        (COMMAND_ATTRIBUTES)(CC_Load * // 0x0157
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2+R_HANDLE)),
#endif
#if (PAD_LIST || CC_Quote)
        (COMMAND_ATTRIBUTES)(CC_Quote * // 0x0158
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_RSA_Decrypt)
        (COMMAND_ATTRIBUTES)(CC_RSA_Decrypt * // 0x0159
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST)
        (COMMAND_ATTRIBUTES)(0), // 0x015A
#endif
#if (PAD_LIST || (CC_HMAC_Start || CC_MAC_Start))
        (COMMAND_ATTRIBUTES)((CC_HMAC_Start || CC_MAC_Start) * // 0x015B
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+R_HANDLE)),
#endif
#if (PAD_LIST || CC_SequenceUpdate)
        (COMMAND_ATTRIBUTES)(CC_SequenceUpdate * // 0x015C
```

```
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_Sign)
        (COMMAND_ATTRIBUTES)(CC_Sign * // 0x015D
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_Unseal)
        (COMMAND_ATTRIBUTES)(CC_Unseal * // 0x015E
            (IS_IMPLEMENTED+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST)
        (COMMAND_ATTRIBUTES)(0), // 0x015F
#endif
#if (PAD_LIST || CC_PolicySigned)
        (COMMAND_ATTRIBUTES)(CC_PolicySigned * // 0x0160
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_ContextLoad)
        (COMMAND_ATTRIBUTES)(CC_ContextLoad * // 0x0161
            (IS_IMPLEMENTED+NO_SESSIONS+R_HANDLE)),
#endif
#if (PAD_LIST || CC_ContextSave)
        (COMMAND_ATTRIBUTES)(CC_ContextSave * // 0x0162
            (IS_IMPLEMENTED+NO_SESSIONS)),
#endif
#if (PAD_LIST || CC_ECDH_KeyGen)
        (COMMAND_ATTRIBUTES)(CC_ECDH_KeyGen * // 0x0163
            (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_EncryptDecrypt)
        (COMMAND_ATTRIBUTES)(CC_EncryptDecrypt * // 0x0164
            (IS_IMPLEMENTED+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_FlushContext)
        (COMMAND_ATTRIBUTES)(CC_FlushContext * // 0x0165
            (IS_IMPLEMENTED+NO_SESSIONS)),
#endif
#if (PAD_LIST)
        (COMMAND_ATTRIBUTES)(0), // 0x0166
#endif
#if (PAD_LIST || CC_LoadExternal)
        (COMMAND_ATTRIBUTES)(CC_LoadExternal * // 0x0167
            (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2+R_HANDLE)),
#endif
#if (PAD_LIST || CC_MakeCredential)
        (COMMAND_ATTRIBUTES)(CC_MakeCredential * // 0x0168
            (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_NV_ReadPublic)
        (COMMAND_ATTRIBUTES)(CC_NV_ReadPublic * // 0x0169
            (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicyAuthorize)
        (COMMAND_ATTRIBUTES)(CC_PolicyAuthorize * // 0x016A
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyAuthValue)
        (COMMAND_ATTRIBUTES)(CC_PolicyAuthValue * // 0x016B
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyCommandCode)
        (COMMAND_ATTRIBUTES)(CC_PolicyCommandCode * // 0x016C
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyCounterTimer)
        (COMMAND_ATTRIBUTES)(CC_PolicyCounterTimer * // 0x016D
```

```
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyCpHash)
        (COMMAND_ATTRIBUTES)(CC_PolicyCpHash * // 0x016E
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyLocality)
        (COMMAND_ATTRIBUTES)(CC_PolicyLocality * // 0x016F
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyNameHash)
        (COMMAND_ATTRIBUTES)(CC_PolicyNameHash * // 0x0170
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyOR)
        (COMMAND_ATTRIBUTES)(CC_PolicyOR * // 0x0171
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyTicket)
        (COMMAND_ATTRIBUTES)(CC_PolicyTicket * // 0x0172
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_ReadPublic)
        (COMMAND_ATTRIBUTES)(CC_ReadPublic * // 0x0173
            (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_RSA_Encrypt)
        (COMMAND_ATTRIBUTES)(CC_RSA_Encrypt * // 0x0174
            (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
#endif
#if (PAD_LIST)
        (COMMAND_ATTRIBUTES)(0), // 0x0175
#endif
#if (PAD_LIST || CC_StartAuthSession)
        (COMMAND_ATTRIBUTES)(CC_StartAuthSession * // 0x0176
            (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2+R_HANDLE)),
#endif
#if (PAD_LIST || CC_VerifySignature)
        (COMMAND_ATTRIBUTES)(CC_VerifySignature * // 0x0177
            (IS_IMPLEMENTED+DECRYPT_2)),
#endif
#if (PAD_LIST || CC_ECC_Parameters)
        (COMMAND_ATTRIBUTES)(CC_ECC_Parameters * // 0x0178
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_FirmwareRead)
        (COMMAND_ATTRIBUTES)(CC_FirmwareRead * // 0x0179
            (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_GetCapability)
        (COMMAND_ATTRIBUTES)(CC_GetCapability * // 0x017A
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_GetRandom)
        (COMMAND_ATTRIBUTES)(CC_GetRandom * // 0x017B
            (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_GetTestResult)
        (COMMAND_ATTRIBUTES)(CC_GetTestResult * // 0x017C
            (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Hash)
        (COMMAND_ATTRIBUTES)(CC_Hash * // 0x017D
            (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PCR_Read)
```

```
        (COMMAND_ATTRIBUTES)(CC_PCR_Read * // 0x017E
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_PolicyPCR)
        (COMMAND_ATTRIBUTES)(CC_PolicyPCR * // 0x017F
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyRestart)
        (COMMAND_ATTRIBUTES)(CC_PolicyRestart * // 0x0180
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_ReadClock)
        (COMMAND_ATTRIBUTES)(CC_ReadClock * // 0x0181
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_PCR_Extend)
        (COMMAND_ATTRIBUTES)(CC_PCR_Extend * // 0x0182
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_PCR_SetAuthValue)
        (COMMAND_ATTRIBUTES)(CC_PCR_SetAuthValue * // 0x0183
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_NV_Certify)
        (COMMAND_ATTRIBUTES)(CC_NV_Certify * // 0x0184
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_EventSequenceComplete)
        (COMMAND_ATTRIBUTES)(CC_EventSequenceComplete * // 0x0185
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER)),
#endif
#if (PAD_LIST || CC_HashSequenceStart)
        (COMMAND_ATTRIBUTES)(CC_HashSequenceStart * // 0x0186
            (IS_IMPLEMENTED+DECRYPT_2+R_HANDLE)),
#endif
#if (PAD_LIST || CC_PolicyPhysicalPresence)
        (COMMAND_ATTRIBUTES)(CC_PolicyPhysicalPresence * // 0x0187
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyDuplicationSelect)
        (COMMAND_ATTRIBUTES)(CC_PolicyDuplicationSelect * // 0x0188
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyGetDigest)
        (COMMAND_ATTRIBUTES)(CC_PolicyGetDigest * // 0x0189
            (IS_IMPLEMENTED+ALLOW_TRIAL+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_TestParms)
        (COMMAND_ATTRIBUTES)(CC_TestParms * // 0x018A
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_Commit)
        (COMMAND_ATTRIBUTES)(CC_Commit * // 0x018B
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicyPassword)
        (COMMAND_ATTRIBUTES)(CC_PolicyPassword * // 0x018C
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_ZGen_2Phase)
        (COMMAND_ATTRIBUTES)(CC_ZGen_2Phase * // 0x018D
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_EC_Ephemeral)
        (COMMAND_ATTRIBUTES)(CC_EC_Ephemeral * // 0x018E
            (IS_IMPLEMENTED+ENCRYPT_2)),
```

```
#endif
#if (PAD_LIST || CC_PolicyNvWritten)
        (COMMAND_ATTRIBUTES)(CC_PolicyNvWritten * // 0x018F
            (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyTemplate)
        (COMMAND_ATTRIBUTES)(CC_PolicyTemplate * // 0x0190
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_CreateLoaded)
        (COMMAND_ATTRIBUTES)(CC_CreateLoaded * // 0x0191
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND+ENCRYPT_2+R_HANDLE)),
#endif
#if (PAD_LIST || CC_PolicyAuthorizeNV)
        (COMMAND_ATTRIBUTES)(CC_PolicyAuthorizeNV * // 0x0192
            (IS_IMPLEMENTED+HANDLE_1_USER+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_EncryptDecrypt2)
        (COMMAND_ATTRIBUTES)(CC_EncryptDecrypt2 * // 0x0193
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_AC_GetCapability)
        (COMMAND_ATTRIBUTES)(CC_AC_GetCapability * // 0x0194
            (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_AC_Send)
        (COMMAND_ATTRIBUTES)(CC_AC_Send * // 0x0195
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_DUP+HANDLE_2_USER)),
#endif
#if (PAD_LIST || CC_Policy_AC_SendSelect)
        (COMMAND_ATTRIBUTES)(CC_Policy_AC_SendSelect * // 0x0196
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_CertifyX509)
        (COMMAND_ATTRIBUTES)(CC_CertifyX509 * // 0x0197
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_ACT_SetTimeout)
        (COMMAND_ATTRIBUTES)(CC_ACT_SetTimeout * // 0x0198
            (IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_ECC_Encrypt)
        (COMMAND_ATTRIBUTES)(CC_ECC_Encrypt * // 0x0199
            (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_ECC_Decrypt)
        (COMMAND_ATTRIBUTES)(CC_ECC_Decrypt * // 0x019A
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicyCapability)
        (COMMAND_ATTRIBUTES)(CC_PolicyCapability * // 0x019B
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyParameters)
        (COMMAND_ATTRIBUTES)(CC_PolicyParameters * // 0x019C
            (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_NV_DefineSpace2)
        (COMMAND_ATTRIBUTES)(CC_NV_DefineSpace2 * // 0x019D
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
#endif
#if (PAD_LIST || CC_NV_ReadPublic2)
        (COMMAND_ATTRIBUTES)(CC_NV_ReadPublic2 * // 0x019E
            (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_SetCapability)
```

```
        (COMMAND_ATTRIBUTES)(CC_SetCapability * // 0x019F
            (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_Vendor_TCG_Test)
        (COMMAND_ATTRIBUTES)(CC_Vendor_TCG_Test * // 0x0000
            (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
#endif
        0
};

#endif // _COMMAND_CODE_ATTRIBUTES_
```

6.12 /tpm/include/private/CommandAttributes.h

```
/*(Auto-generated)
 *  Created by TpmStructures; Version 4.4 Mar 26, 2019
 *  Date: Aug 30, 2019  Time: 02:11:52PM
 */

// The attributes defined in this file are produced by the parser that
// creates the structure definitions from Part 3. The attributes are defined
// in that parser and should track the attributes being tested in
// CommandCodeAttributes.c. Generally, when an attribute is added to this list,
// new code will be needed in CommandCodeAttributes.c to test it.

#ifndef COMMAND_ATTRIBUTES_H
#define COMMAND_ATTRIBUTES_H

typedef UINT16 COMMAND_ATTRIBUTES;
#define NOT_IMPLEMENTED (COMMAND_ATTRIBUTES)(0)
#define ENCRYPT_2       ((COMMAND_ATTRIBUTES)1 << 0)
#define ENCRYPT_4       ((COMMAND_ATTRIBUTES)1 << 1)
#define DECRYPT_2       ((COMMAND_ATTRIBUTES)1 << 2)
#define DECRYPT_4       ((COMMAND_ATTRIBUTES)1 << 3)
#define HANDLE_1_USER   ((COMMAND_ATTRIBUTES)1 << 4)
#define HANDLE_1_ADMIN  ((COMMAND_ATTRIBUTES)1 << 5)
#define HANDLE_1_DUP    ((COMMAND_ATTRIBUTES)1 << 6)
#define HANDLE_2_USER   ((COMMAND_ATTRIBUTES)1 << 7)
#define PP_COMMAND      ((COMMAND_ATTRIBUTES)1 << 8)
#define IS_IMPLEMENTED  ((COMMAND_ATTRIBUTES)1 << 9)
#define NO_SESSIONS     ((COMMAND_ATTRIBUTES)1 << 10)
#define NV_COMMAND      ((COMMAND_ATTRIBUTES)1 << 11)
#define PP_REQUIRED     ((COMMAND_ATTRIBUTES)1 << 12)
#define R_HANDLE        ((COMMAND_ATTRIBUTES)1 << 13)
#define ALLOW_TRIAL     ((COMMAND_ATTRIBUTES)1 << 14)

#endif  // COMMAND_ATTRIBUTES_H
```

6.13 /tpm/include/private/CommandDispatchData.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT
// clang-format off

// This file should only be included by CommandCodeAttributes.c
#ifdef _COMMAND_TABLE_DISPATCH_

// Define the stop value
#define END_OF_LIST     0xff
#define ADD_FLAG        0x80

// These macros provide some variability in how the data is encoded. They also
// make the lines a little shorter. :-)
// When TABLE_DRIVEN_MARSHAL is 'NO', the un/marshaling of parameters uses
// calls to the function that does the type-specific un/marshaling. When
```

```
// TABLE_DRIVEN_MARSHAL is 'YES', the un/marshaling of parameters calls the
// singular code with a value that is the offset of the data descriptor of the
// type.
#if TABLE_DRIVEN_MARSHAL
#   define UNMARSHAL_DISPATCH(name)    (marshalIndex_t)name##_MARSHAL_REF
#   define MARSHAL_DISPATCH(name)      (marshalIndex_t)name##_MARSHAL_REF
#   define _UNMARSHAL_T_     marshalIndex_t
#   define _MARSHAL_T_       marshalIndex_t
#else
#   define UNMARSHAL_DISPATCH(name)    (UNMARSHAL_t)name##_Unmarshal
#   define MARSHAL_DISPATCH(name)      (MARSHAL_t)name##_Marshal
#   define _UNMARSHAL_T_     UNMARSHAL_t
#   define _MARSHAL_T_       MARSHAL_t
#endif

// The unmarshalArray contains the dispatch functions for the unmarshaling
// code. The defines in this array are used to make it easier to cross
// reference the unmarshaling values in the types array of each command

const _UNMARSHAL_T_ unmarshalArray[] = {
#define TPMI_DH_CONTEXT_H_UNMARSHAL             0
            UNMARSHAL_DISPATCH(TPMI_DH_CONTEXT),
#define TPMI_RH_AC_H_UNMARSHAL                  (TPMI_DH_CONTEXT_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_AC),
#define TPMI_RH_ACT_H_UNMARSHAL                 (TPMI_RH_AC_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_ACT),
#define TPMI_RH_CLEAR_H_UNMARSHAL               (TPMI_RH_ACT_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_CLEAR),
#define TPMI_RH_HIERARCHY_AUTH_H_UNMARSHAL      (TPMI_RH_CLEAR_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_HIERARCHY_AUTH),
#define TPMI_RH_HIERARCHY_POLICY_H_UNMARSHAL                                    \
            (TPMI_RH_HIERARCHY_AUTH_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_HIERARCHY_POLICY),
#define TPMI_RH_BASE_HIERARCHY_H_UNMARSHAL                                      \
            (TPMI_RH_HIERARCHY_POLICY_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_BASE_HIERARCHY),
#define TPMI_RH_LOCKOUT_H_UNMARSHAL       (TPMI_RH_BASE_HIERARCHY_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_LOCKOUT),
#define TPMI_RH_NV_AUTH_H_UNMARSHAL             (TPMI_RH_LOCKOUT_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_NV_AUTH),
#define TPMI_RH_NV_DEFINED_INDEX_H_UNMARSHAL    (TPMI_RH_NV_AUTH_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_NV_DEFINED_INDEX),
#define TPMI_RH_NV_INDEX_H_UNMARSHAL      (TPMI_RH_NV_DEFINED_INDEX_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_NV_INDEX),
#define TPMI_RH_PLATFORM_H_UNMARSHAL            (TPMI_RH_NV_INDEX_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_PLATFORM),
#define TPMI_RH_PROVISION_H_UNMARSHAL           (TPMI_RH_PLATFORM_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_PROVISION),
#define TPMI_SH_HMAC_H_UNMARSHAL                (TPMI_RH_PROVISION_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_SH_HMAC),
#define TPMI_SH_POLICY_H_UNMARSHAL              (TPMI_SH_HMAC_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_SH_POLICY),
// HANDLE_FIRST_FLAG_TYPE is the first handle that needs a flag when called.
#define HANDLE_FIRST_FLAG_TYPE                  (TPMI_SH_POLICY_H_UNMARSHAL + 1)
#define TPMI_DH_ENTITY_H_UNMARSHAL              (TPMI_SH_POLICY_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_DH_ENTITY),
#define TPMI_DH_OBJECT_H_UNMARSHAL              (TPMI_DH_ENTITY_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_DH_OBJECT),
#define TPMI_DH_PARENT_H_UNMARSHAL              (TPMI_DH_OBJECT_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_DH_PARENT),
#define TPMI_DH_PCR_H_UNMARSHAL                 (TPMI_DH_PARENT_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_DH_PCR),
#define TPMI_RH_ENDORSEMENT_H_UNMARSHAL         (TPMI_DH_PCR_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_ENDORSEMENT),
#define TPMI_RH_HIERARCHY_H_UNMARSHAL       (TPMI_RH_ENDORSEMENT_H_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_HIERARCHY),
```

```
// PARAMETER_FIRST_TYPE marks the end of the handle list.
#define PARAMETER_FIRST_TYPE                    (TPMI_RH_HIERARCHY_H_UNMARSHAL + 1)
#define TPM_AT_P_UNMARSHAL                      (TPMI_RH_HIERARCHY_H_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM_AT),
#define TPM_CAP_P_UNMARSHAL                     (TPM_AT_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM_CAP),
#define TPM_CLOCK_ADJUST_P_UNMARSHAL            (TPM_CAP_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM_CLOCK_ADJUST),
#define TPM_EO_P_UNMARSHAL                      (TPM_CLOCK_ADJUST_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM_EO),
#define TPM_SE_P_UNMARSHAL                      (TPM_EO_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM_SE),
#define TPM_SU_P_UNMARSHAL                      (TPM_SE_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM_SU),
#define TPM2B_DATA_P_UNMARSHAL                  (TPM_SU_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_DATA),
#define TPM2B_DIGEST_P_UNMARSHAL                (TPM2B_DATA_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_DIGEST),
#define TPM2B_ECC_PARAMETER_P_UNMARSHAL         (TPM2B_DIGEST_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_ECC_PARAMETER),
#define TPM2B_ECC_POINT_P_UNMARSHAL           (TPM2B_ECC_PARAMETER_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_ECC_POINT),
#define TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL      (TPM2B_ECC_POINT_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_ENCRYPTED_SECRET),
#define TPM2B_EVENT_P_UNMARSHAL             (TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_EVENT),
#define TPM2B_ID_OBJECT_P_UNMARSHAL             (TPM2B_EVENT_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_ID_OBJECT),
#define TPM2B_IV_P_UNMARSHAL                    (TPM2B_ID_OBJECT_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_IV),
#define TPM2B_MAX_BUFFER_P_UNMARSHAL            (TPM2B_IV_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_MAX_BUFFER),
#define TPM2B_MAX_NV_BUFFER_P_UNMARSHAL         (TPM2B_MAX_BUFFER_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_MAX_NV_BUFFER),
#define TPM2B_NAME_P_UNMARSHAL                (TPM2B_MAX_NV_BUFFER_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_NAME),
#define TPM2B_NV_PUBLIC_P_UNMARSHAL             (TPM2B_NAME_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_NV_PUBLIC),
#define TPM2B_NV_PUBLIC_2_P_UNMARSHAL           (TPM2B_NV_PUBLIC_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_NV_PUBLIC_2),
#define TPM2B_PRIVATE_P_UNMARSHAL               (TPM2B_NV_PUBLIC_2_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_PRIVATE),
#define TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL        (TPM2B_PRIVATE_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_PUBLIC_KEY_RSA),
#define TPM2B_SENSITIVE_P_UNMARSHAL           (TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_SENSITIVE),
#define TPM2B_SENSITIVE_CREATE_P_UNMARSHAL      (TPM2B_SENSITIVE_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_SENSITIVE_CREATE),
#define TPM2B_SENSITIVE_DATA_P_UNMARSHAL    (TPM2B_SENSITIVE_CREATE_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_SENSITIVE_DATA),
#define TPM2B_SET_CAPABILITY_DATA_P_UNMARSHAL   (TPM2B_SENSITIVE_DATA_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_SET_CAPABILITY_DATA),
#define TPM2B_TEMPLATE_P_UNMARSHAL            (TPM2B_SET_CAPABILITY_DATA_P_UNMARSHAL +
1)
        UNMARSHAL_DISPATCH(TPM2B_TEMPLATE),
#define TPM2B_TIMEOUT_P_UNMARSHAL               (TPM2B_TEMPLATE_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPM2B_TIMEOUT),
#define TPMI_DH_CONTEXT_P_UNMARSHAL             (TPM2B_TIMEOUT_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPMI_DH_CONTEXT),
#define TPMI_DH_PERSISTENT_P_UNMARSHAL          (TPMI_DH_CONTEXT_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPMI_DH_PERSISTENT),
#define TPMI_YES_NO_P_UNMARSHAL                 (TPMI_DH_PERSISTENT_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPMI_YES_NO),
#define TPML_ALG_P_UNMARSHAL                    (TPMI_YES_NO_P_UNMARSHAL + 1)
        UNMARSHAL_DISPATCH(TPML_ALG),
#define TPML_CC_P_UNMARSHAL                     (TPML_ALG_P_UNMARSHAL + 1)
```

```
            UNMARSHAL_DISPATCH(TPML_CC),
#define TPML_DIGEST_P_UNMARSHAL                 (TPML_CC_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPML_DIGEST),
#define TPML_DIGEST_VALUES_P_UNMARSHAL          (TPML_DIGEST_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPML_DIGEST_VALUES),
#define TPML_PCR_SELECTION_P_UNMARSHAL          (TPML_DIGEST_VALUES_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPML_PCR_SELECTION),
#define TPMS_CONTEXT_P_UNMARSHAL                (TPML_PCR_SELECTION_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMS_CONTEXT),
#define TPMT_PUBLIC_PARMS_P_UNMARSHAL           (TPMS_CONTEXT_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_PUBLIC_PARMS),
#define TPMT_TK_AUTH_P_UNMARSHAL                (TPMT_PUBLIC_PARMS_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_TK_AUTH),
#define TPMT_TK_CREATION_P_UNMARSHAL            (TPMT_TK_AUTH_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_TK_CREATION),
#define TPMT_TK_HASHCHECK_P_UNMARSHAL           (TPMT_TK_CREATION_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_TK_HASHCHECK),
#define TPMT_TK_VERIFIED_P_UNMARSHAL            (TPMT_TK_HASHCHECK_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_TK_VERIFIED),
#define UINT16_P_UNMARSHAL                      (TPMT_TK_VERIFIED_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(UINT16),
#define UINT32_P_UNMARSHAL                      (UINT16_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(UINT32),
#define UINT64_P_UNMARSHAL                      (UINT32_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(UINT64),
#define UINT8_P_UNMARSHAL                       (UINT64_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(UINT8),
// PARAMETER_FIRST_FLAG_TYPE is the first parameter to need a flag.
#define PARAMETER_FIRST_FLAG_TYPE               (UINT8_P_UNMARSHAL + 1)
#define TPM2B_PUBLIC_P_UNMARSHAL                (UINT8_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPM2B_PUBLIC),
#define TPMI_ALG_CIPHER_MODE_P_UNMARSHAL        (TPM2B_PUBLIC_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_ALG_CIPHER_MODE),
#define TPMI_ALG_HASH_P_UNMARSHAL             (TPMI_ALG_CIPHER_MODE_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_ALG_HASH),
#define TPMI_ALG_MAC_SCHEME_P_UNMARSHAL         (TPMI_ALG_HASH_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_ALG_MAC_SCHEME),
#define TPMI_DH_PCR_P_UNMARSHAL               (TPMI_ALG_MAC_SCHEME_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_DH_PCR),
#define TPMI_ECC_CURVE_P_UNMARSHAL              (TPMI_DH_PCR_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_ECC_CURVE),
#define TPMI_ECC_KEY_EXCHANGE_P_UNMARSHAL       (TPMI_ECC_CURVE_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_ECC_KEY_EXCHANGE),
#define TPMI_RH_ENABLES_P_UNMARSHAL           (TPMI_ECC_KEY_EXCHANGE_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_ENABLES),
#define TPMI_RH_HIERARCHY_P_UNMARSHAL           (TPMI_RH_ENABLES_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMI_RH_HIERARCHY),
#define TPMT_KDF_SCHEME_P_UNMARSHAL             (TPMI_RH_HIERARCHY_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_KDF_SCHEME),
#define TPMT_RSA_DECRYPT_P_UNMARSHAL            (TPMT_KDF_SCHEME_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_RSA_DECRYPT),
#define TPMT_SIG_SCHEME_P_UNMARSHAL             (TPMT_RSA_DECRYPT_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_SIG_SCHEME),
#define TPMT_SIGNATURE_P_UNMARSHAL              (TPMT_SIG_SCHEME_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_SIGNATURE),
#define TPMT_SYM_DEF_P_UNMARSHAL                (TPMT_SIGNATURE_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_SYM_DEF),
#define TPMT_SYM_DEF_OBJECT_P_UNMARSHAL         (TPMT_SYM_DEF_P_UNMARSHAL + 1)
            UNMARSHAL_DISPATCH(TPMT_SYM_DEF_OBJECT)
// PARAMETER_LAST_TYPE is the index of the last command parameter.
#define PARAMETER_LAST_TYPE                     (TPMT_SYM_DEF_OBJECT_P_UNMARSHAL)
};

// The marshalArray contains the dispatch functions for the marshaling code.
// The defines in this array are used to make it easier to cross reference the
// marshaling values in the types array of each command
```

```
const _MARSHAL_T_ marshalArray[] = {
#define UINT32_H_MARSHAL                        0
            MARSHAL_DISPATCH(UINT32),
// RESPONSE_PARAMETER_FIRST_TYPE marks the end of the response handles.
#define RESPONSE_PARAMETER_FIRST_TYPE           (UINT32_H_MARSHAL + 1)
#define TPM2B_ATTEST_P_MARSHAL                  (UINT32_H_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_ATTEST),
#define TPM2B_CREATION_DATA_P_MARSHAL           (TPM2B_ATTEST_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_CREATION_DATA),
#define TPM2B_DATA_P_MARSHAL                    (TPM2B_CREATION_DATA_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_DATA),
#define TPM2B_DIGEST_P_MARSHAL                  (TPM2B_DATA_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_DIGEST),
#define TPM2B_ECC_POINT_P_MARSHAL               (TPM2B_DIGEST_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_ECC_POINT),
#define TPM2B_ENCRYPTED_SECRET_P_MARSHAL        (TPM2B_ECC_POINT_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_ENCRYPTED_SECRET),
#define TPM2B_ID_OBJECT_P_MARSHAL             (TPM2B_ENCRYPTED_SECRET_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_ID_OBJECT),
#define TPM2B_IV_P_MARSHAL                      (TPM2B_ID_OBJECT_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_IV),
#define TPM2B_MAX_BUFFER_P_MARSHAL              (TPM2B_IV_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_MAX_BUFFER),
#define TPM2B_MAX_NV_BUFFER_P_MARSHAL           (TPM2B_MAX_BUFFER_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_MAX_NV_BUFFER),
#define TPM2B_NAME_P_MARSHAL                    (TPM2B_MAX_NV_BUFFER_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_NAME),
#define TPM2B_NV_PUBLIC_P_MARSHAL               (TPM2B_NAME_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_NV_PUBLIC),
#define TPM2B_NV_PUBLIC_2_P_MARSHAL             (TPM2B_NV_PUBLIC_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_NV_PUBLIC_2),
#define TPM2B_PRIVATE_P_MARSHAL                 (TPM2B_NV_PUBLIC_2_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_PRIVATE),
#define TPM2B_PUBLIC_P_MARSHAL                  (TPM2B_PRIVATE_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_PUBLIC),
#define TPM2B_PUBLIC_KEY_RSA_P_MARSHAL          (TPM2B_PUBLIC_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_PUBLIC_KEY_RSA),
#define TPM2B_SENSITIVE_DATA_P_MARSHAL          (TPM2B_PUBLIC_KEY_RSA_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_SENSITIVE_DATA),
#define TPM2B_TIMEOUT_P_MARSHAL                 (TPM2B_SENSITIVE_DATA_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPM2B_TIMEOUT),
#define TPML_AC_CAPABILITIES_P_MARSHAL          (TPM2B_TIMEOUT_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPML_AC_CAPABILITIES),
#define TPML_ALG_P_MARSHAL                      (TPML_AC_CAPABILITIES_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPML_ALG),
#define TPML_DIGEST_P_MARSHAL                   (TPML_ALG_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPML_DIGEST),
#define TPML_DIGEST_VALUES_P_MARSHAL            (TPML_DIGEST_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPML_DIGEST_VALUES),
#define TPML_PCR_SELECTION_P_MARSHAL            (TPML_DIGEST_VALUES_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPML_PCR_SELECTION),
#define TPMS_AC_OUTPUT_P_MARSHAL                (TPML_PCR_SELECTION_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMS_AC_OUTPUT),
#define TPMS_ALGORITHM_DETAIL_ECC_P_MARSHAL     (TPMS_AC_OUTPUT_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMS_ALGORITHM_DETAIL_ECC),
#define TPMS_CAPABILITY_DATA_P_MARSHAL      (TPMS_ALGORITHM_DETAIL_ECC_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMS_CAPABILITY_DATA),
#define TPMS_CONTEXT_P_MARSHAL                  (TPMS_CAPABILITY_DATA_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMS_CONTEXT),
#define TPMS_TIME_INFO_P_MARSHAL                (TPMS_CONTEXT_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMS_TIME_INFO),
#define TPMT_HA_P_MARSHAL                       (TPMS_TIME_INFO_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMT_HA),
#define TPMT_SIGNATURE_P_MARSHAL                (TPMT_HA_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMT_SIGNATURE),
#define TPMT_TK_AUTH_P_MARSHAL                  (TPMT_SIGNATURE_P_MARSHAL + 1)
```

```
            MARSHAL_DISPATCH(TPMT_TK_AUTH),
#define TPMT_TK_CREATION_P_MARSHAL              (TPMT_TK_AUTH_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMT_TK_CREATION),
#define TPMT_TK_HASHCHECK_P_MARSHAL             (TPMT_TK_CREATION_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMT_TK_HASHCHECK),
#define TPMT_TK_VERIFIED_P_MARSHAL              (TPMT_TK_HASHCHECK_P_MARSHAL + 1)
            MARSHAL_DISPATCH(TPMT_TK_VERIFIED),
#define UINT16_P_MARSHAL                        (TPMT_TK_VERIFIED_P_MARSHAL + 1)
            MARSHAL_DISPATCH(UINT16),
#define UINT32_P_MARSHAL                        (UINT16_P_MARSHAL + 1)
            MARSHAL_DISPATCH(UINT32),
#define UINT8_P_MARSHAL                         (UINT32_P_MARSHAL + 1)
            MARSHAL_DISPATCH(UINT8)
// RESPONSE_PARAMETER_LAST_TYPE is the index of the last response parameter.
#define RESPONSE_PARAMETER_LAST_TYPE            (UINT8_P_MARSHAL)
};

// This list of aliases allows the types in the _COMMAND_DESCRIPTOR_t to match
// the types in the command/response templates of part 3.
#define TPM2B_NONCE_P_UNMARSHAL                 TPM2B_DIGEST_P_UNMARSHAL
#define TPM2B_AUTH_P_UNMARSHAL                  TPM2B_DIGEST_P_UNMARSHAL
#define TPM2B_OPERAND_P_UNMARSHAL               TPM2B_DIGEST_P_UNMARSHAL
#define INT32_P_UNMARSHAL                       UINT32_P_UNMARSHAL
#define TPM_CC_P_UNMARSHAL                      UINT32_P_UNMARSHAL
#define TPMA_LOCALITY_P_UNMARSHAL               UINT8_P_UNMARSHAL
#define TPMI_SH_AUTH_SESSION_H_MARSHAL          UINT32_H_MARSHAL
#define TPM_HANDLE_H_MARSHAL                    UINT32_H_MARSHAL
#define TPMI_DH_OBJECT_H_MARSHAL                UINT32_H_MARSHAL
#define TPMI_DH_CONTEXT_H_MARSHAL               UINT32_H_MARSHAL
#define TPM2B_NONCE_P_MARSHAL                   TPM2B_DIGEST_P_MARSHAL
#define TPM_RC_P_MARSHAL                        UINT32_P_MARSHAL
#define TPMI_YES_NO_P_MARSHAL                   UINT8_P_MARSHAL


// Per-command un/marshaling tables

#if       CC_Startup
#include    "Startup_fp.h"

typedef TPM_RC (Startup_Entry)(
    Startup_In*                 in
);


typedef const struct
{
    Startup_Entry               *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} Startup_COMMAND_DESCRIPTOR_t;

Startup_COMMAND_DESCRIPTOR_t _StartupData = {
    /* entry         */         &TPM2_Startup,
    /* inSize        */         (UINT16)(sizeof(Startup_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(Startup_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPM_SU_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _StartupDataAddress (&_StartupData)
#else
```

```
#define _StartupDataAddress 0
#endif // CC_Startup

#if       CC_Shutdown
#include    "Shutdown_fp.h"

typedef TPM_RC (Shutdown_Entry)(
    Shutdown_In*                in
);


typedef const struct
{
    Shutdown_Entry              *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} Shutdown_COMMAND_DESCRIPTOR_t;

Shutdown_COMMAND_DESCRIPTOR_t _ShutdownData = {
    /* entry         */         &TPM2_Shutdown,
    /* inSize        */         (UINT16)(sizeof(Shutdown_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(Shutdown_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPM_SU_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _ShutdownDataAddress (&_ShutdownData)
#else
#define _ShutdownDataAddress 0
#endif // CC_Shutdown

#if       CC_SelfTest
#include    "SelfTest_fp.h"

typedef TPM_RC (SelfTest_Entry)(
    SelfTest_In*                in
);


typedef const struct
{
    SelfTest_Entry              *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} SelfTest_COMMAND_DESCRIPTOR_t;

SelfTest_COMMAND_DESCRIPTOR_t _SelfTestData = {
    /* entry         */         &TPM2_SelfTest,
    /* inSize        */         (UINT16)(sizeof(SelfTest_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(SelfTest_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_YES_NO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _SelfTestDataAddress (&_SelfTestData)
#else
```

```
#define _SelfTestDataAddress 0
#endif // CC_SelfTest

#if       CC_IncrementalSelfTest
#include    "IncrementalSelfTest_fp.h"

typedef TPM_RC (IncrementalSelfTest_Entry)(
    IncrementalSelfTest_In*     in,
    IncrementalSelfTest_Out*    out
);


typedef const struct
{
    IncrementalSelfTest_Entry   *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} IncrementalSelfTest_COMMAND_DESCRIPTOR_t;

IncrementalSelfTest_COMMAND_DESCRIPTOR_t _IncrementalSelfTestData = {
    /* entry         */         &TPM2_IncrementalSelfTest,
    /* inSize        */         (UINT16)(sizeof(IncrementalSelfTest_In)),
    /* outSize       */         (UINT16)(sizeof(IncrementalSelfTest_Out)),
    /* offsetOfTypes */         offsetof(IncrementalSelfTest_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPML_ALG_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPML_ALG_P_MARSHAL,
                                 END_OF_LIST}
};

#define _IncrementalSelfTestDataAddress (&_IncrementalSelfTestData)
#else
#define _IncrementalSelfTestDataAddress 0
#endif // CC_IncrementalSelfTest

#if       CC_GetTestResult
#include    "GetTestResult_fp.h"

typedef TPM_RC (GetTestResult_Entry)(
    GetTestResult_Out*          out
);


typedef const struct
{
    GetTestResult_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} GetTestResult_COMMAND_DESCRIPTOR_t;

GetTestResult_COMMAND_DESCRIPTOR_t _GetTestResultData = {
    /* entry         */         &TPM2_GetTestResult,
    /* inSize        */         0,
    /* outSize       */         (UINT16)(sizeof(GetTestResult_Out)),
    /* offsetOfTypes */         offsetof(GetTestResult_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(GetTestResult_Out, testResult))},
    /* types         */         {END_OF_LIST,
                                 TPM2B_MAX_BUFFER_P_MARSHAL,
                                 TPM_RC_P_MARSHAL,
```

```
                                    END_OF_LIST}
};

#define _GetTestResultDataAddress (&_GetTestResultData)
#else
#define _GetTestResultDataAddress 0
#endif // CC_GetTestResult

#if       CC_StartAuthSession
#include    "StartAuthSession_fp.h"

typedef TPM_RC (StartAuthSession_Entry)(
    StartAuthSession_In*        in,
    StartAuthSession_Out*       out
);


typedef const struct
{
    StartAuthSession_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[7];
    BYTE                        types[11];
} StartAuthSession_COMMAND_DESCRIPTOR_t;

StartAuthSession_COMMAND_DESCRIPTOR_t _StartAuthSessionData = {
    /* entry          */        &TPM2_StartAuthSession,
    /* inSize         */        (UINT16)(sizeof(StartAuthSession_In)),
    /* outSize        */        (UINT16)(sizeof(StartAuthSession_Out)),
    /* offsetOfTypes */         offsetof(StartAuthSession_COMMAND_DESCRIPTOR_t,
types),
    /* offsets        */        {(UINT16)(offsetof(StartAuthSession_In, bind)),
                                 (UINT16)(offsetof(StartAuthSession_In, nonceCaller)),
                                 (UINT16)(offsetof(StartAuthSession_In,
encryptedSalt)),
                                 (UINT16)(offsetof(StartAuthSession_In, sessionType)),
                                 (UINT16)(offsetof(StartAuthSession_In, symmetric)),
                                 (UINT16)(offsetof(StartAuthSession_In, authHash)),
                                 (UINT16)(offsetof(StartAuthSession_Out, nonceTPM))},
    /* types          */        {TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPMI_DH_ENTITY_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_NONCE_P_UNMARSHAL,
                                 TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
                                 TPM_SE_P_UNMARSHAL,
                                 TPMT_SYM_DEF_P_UNMARSHAL + ADD_FLAG,
                                 TPMI_ALG_HASH_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMI_SH_AUTH_SESSION_H_MARSHAL,
                                 TPM2B_NONCE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _StartAuthSessionDataAddress (&_StartAuthSessionData)
#else
#define _StartAuthSessionDataAddress 0
#endif // CC_StartAuthSession

#if       CC_PolicyRestart
#include    "PolicyRestart_fp.h"

typedef TPM_RC (PolicyRestart_Entry)(
    PolicyRestart_In*           in
);
```

```
typedef const struct
{
    PolicyRestart_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} PolicyRestart_COMMAND_DESCRIPTOR_t;

PolicyRestart_COMMAND_DESCRIPTOR_t _PolicyRestartData = {
    /* entry         */         &TPM2_PolicyRestart,
    /* inSize        */         (UINT16)(sizeof(PolicyRestart_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyRestart_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyRestartDataAddress (&_PolicyRestartData)
#else
#define _PolicyRestartDataAddress 0
#endif // CC_PolicyRestart

#if       CC_Create
#include    "Create_fp.h"

typedef TPM_RC (Create_Entry)(
    Create_In*                  in,
    Create_Out*                 out
);


typedef const struct
{
    Create_Entry                *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[8];
    BYTE                        types[12];
} Create_COMMAND_DESCRIPTOR_t;

Create_COMMAND_DESCRIPTOR_t _CreateData = {
    /* entry         */         &TPM2_Create,
    /* inSize        */         (UINT16)(sizeof(Create_In)),
    /* outSize       */         (UINT16)(sizeof(Create_Out)),
    /* offsetOfTypes */         offsetof(Create_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(Create_In, inSensitive)),
                                 (UINT16)(offsetof(Create_In, inPublic)),
                                 (UINT16)(offsetof(Create_In, outsideInfo)),
                                 (UINT16)(offsetof(Create_In, creationPCR)),
                                 (UINT16)(offsetof(Create_Out, outPublic)),
                                 (UINT16)(offsetof(Create_Out, creationData)),
                                 (UINT16)(offsetof(Create_Out, creationHash)),
                                 (UINT16)(offsetof(Create_Out, creationTicket))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_SENSITIVE_CREATE_P_UNMARSHAL,
                                 TPM2B_PUBLIC_P_UNMARSHAL,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPML_PCR_SELECTION_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_PRIVATE_P_MARSHAL,
                                 TPM2B_PUBLIC_P_MARSHAL,
```

```
                                    TPM2B_CREATION_DATA_P_MARSHAL,
                                    TPM2B_DIGEST_P_MARSHAL,
                                    TPMT_TK_CREATION_P_MARSHAL,
                                    END_OF_LIST}
};

#define _CreateDataAddress (&_CreateData)
#else
#define _CreateDataAddress 0
#endif // CC_Create

#if       CC_Load
#include    "Load_fp.h"

typedef TPM_RC (Load_Entry)(
    Load_In*                    in,
    Load_Out*                   out
);


typedef const struct
{
    Load_Entry                  *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} Load_COMMAND_DESCRIPTOR_t;

Load_COMMAND_DESCRIPTOR_t _LoadData = {
    /* entry         */         &TPM2_Load,
    /* inSize        */         (UINT16)(sizeof(Load_In)),
    /* outSize       */         (UINT16)(sizeof(Load_Out)),
    /* offsetOfTypes */         offsetof(Load_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(Load_In, inPrivate)),
                                 (UINT16)(offsetof(Load_In, inPublic)),
                                 (UINT16)(offsetof(Load_Out, name))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_PRIVATE_P_UNMARSHAL,
                                 TPM2B_PUBLIC_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM_HANDLE_H_MARSHAL,
                                 TPM2B_NAME_P_MARSHAL,
                                 END_OF_LIST}
};

#define _LoadDataAddress (&_LoadData)
#else
#define _LoadDataAddress 0
#endif // CC_Load

#if       CC_LoadExternal
#include    "LoadExternal_fp.h"

typedef TPM_RC (LoadExternal_Entry)(
    LoadExternal_In*            in,
    LoadExternal_Out*           out
);


typedef const struct
{
    LoadExternal_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
```

```
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} LoadExternal_COMMAND_DESCRIPTOR_t;

LoadExternal_COMMAND_DESCRIPTOR_t _LoadExternalData = {
    /* entry         */         &TPM2_LoadExternal,
    /* inSize        */         (UINT16)(sizeof(LoadExternal_In)),
    /* outSize       */         (UINT16)(sizeof(LoadExternal_Out)),
    /* offsetOfTypes */         offsetof(LoadExternal_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(LoadExternal_In, inPublic)),
                                 (UINT16)(offsetof(LoadExternal_In, hierarchy)),
                                 (UINT16)(offsetof(LoadExternal_Out, name))},
    /* types         */         {TPM2B_SENSITIVE_P_UNMARSHAL,
                                 TPM2B_PUBLIC_P_UNMARSHAL + ADD_FLAG,
                                 TPMI_RH_HIERARCHY_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM_HANDLE_H_MARSHAL,
                                 TPM2B_NAME_P_MARSHAL,
                                 END_OF_LIST}
};

#define _LoadExternalDataAddress (&_LoadExternalData)
#else
#define _LoadExternalDataAddress 0
#endif // CC_LoadExternal

#if       CC_ReadPublic
#include    "ReadPublic_fp.h"

typedef TPM_RC (ReadPublic_Entry)(
    ReadPublic_In*              in,
    ReadPublic_Out*             out
);


typedef const struct
{
    ReadPublic_Entry            *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} ReadPublic_COMMAND_DESCRIPTOR_t;

ReadPublic_COMMAND_DESCRIPTOR_t _ReadPublicData = {
    /* entry         */         &TPM2_ReadPublic,
    /* inSize        */         (UINT16)(sizeof(ReadPublic_In)),
    /* outSize       */         (UINT16)(sizeof(ReadPublic_Out)),
    /* offsetOfTypes */         offsetof(ReadPublic_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ReadPublic_Out, name)),
                                 (UINT16)(offsetof(ReadPublic_Out, qualifiedName))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_PUBLIC_P_MARSHAL,
                                 TPM2B_NAME_P_MARSHAL,
                                 TPM2B_NAME_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ReadPublicDataAddress (&_ReadPublicData)
#else
#define _ReadPublicDataAddress 0
#endif // CC_ReadPublic
```

```
#if       CC_ActivateCredential
#include    "ActivateCredential_fp.h"

typedef TPM_RC (ActivateCredential_Entry)(
    ActivateCredential_In*      in,
    ActivateCredential_Out*     out
);


typedef const struct
{
    ActivateCredential_Entry    *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} ActivateCredential_COMMAND_DESCRIPTOR_t;

ActivateCredential_COMMAND_DESCRIPTOR_t _ActivateCredentialData = {
    /* entry         */         &TPM2_ActivateCredential,
    /* inSize        */         (UINT16)(sizeof(ActivateCredential_In)),
    /* outSize       */         (UINT16)(sizeof(ActivateCredential_Out)),
    /* offsetOfTypes */         offsetof(ActivateCredential_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(ActivateCredential_In, keyHandle)),
                                 (UINT16)(offsetof(ActivateCredential_In,
credentialBlob)),
                                 (UINT16)(offsetof(ActivateCredential_In, secret))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_ID_OBJECT_P_UNMARSHAL,
                                 TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ActivateCredentialDataAddress (&_ActivateCredentialData)
#else
#define _ActivateCredentialDataAddress 0
#endif // CC_ActivateCredential

#if       CC_MakeCredential
#include    "MakeCredential_fp.h"

typedef TPM_RC (MakeCredential_Entry)(
    MakeCredential_In*          in,
    MakeCredential_Out*         out
);


typedef const struct
{
    MakeCredential_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} MakeCredential_COMMAND_DESCRIPTOR_t;

MakeCredential_COMMAND_DESCRIPTOR_t _MakeCredentialData = {
    /* entry         */         &TPM2_MakeCredential,
    /* inSize        */         (UINT16)(sizeof(MakeCredential_In)),
    /* outSize       */         (UINT16)(sizeof(MakeCredential_Out)),
```

```
    /* offsetOfTypes */         offsetof(MakeCredential_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(MakeCredential_In, credential)),
                                 (UINT16)(offsetof(MakeCredential_In, objectName)),
                                 (UINT16)(offsetof(MakeCredential_Out, secret))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPM2B_NAME_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ID_OBJECT_P_MARSHAL,
                                 TPM2B_ENCRYPTED_SECRET_P_MARSHAL,
                                 END_OF_LIST}
};

#define _MakeCredentialDataAddress (&_MakeCredentialData)
#else
#define _MakeCredentialDataAddress 0
#endif // CC_MakeCredential

#if       CC_Unseal
#include    "Unseal_fp.h"

typedef TPM_RC (Unseal_Entry)(
    Unseal_In*                  in,
    Unseal_Out*                 out
);


typedef const struct
{
    Unseal_Entry                *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} Unseal_COMMAND_DESCRIPTOR_t;

Unseal_COMMAND_DESCRIPTOR_t _UnsealData = {
    /* entry         */         &TPM2_Unseal,
    /* inSize        */         (UINT16)(sizeof(Unseal_In)),
    /* outSize       */         (UINT16)(sizeof(Unseal_Out)),
    /* offsetOfTypes */         offsetof(Unseal_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_SENSITIVE_DATA_P_MARSHAL,
                                 END_OF_LIST}
};

#define _UnsealDataAddress (&_UnsealData)
#else
#define _UnsealDataAddress 0
#endif // CC_Unseal

#if       CC_ObjectChangeAuth
#include    "ObjectChangeAuth_fp.h"

typedef TPM_RC (ObjectChangeAuth_Entry)(
    ObjectChangeAuth_In*        in,
    ObjectChangeAuth_Out*       out
);


typedef const struct
{
    ObjectChangeAuth_Entry      *entry;
    UINT16                      inSize;
```

```
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} ObjectChangeAuth_COMMAND_DESCRIPTOR_t;

ObjectChangeAuth_COMMAND_DESCRIPTOR_t _ObjectChangeAuthData = {
    /* entry         */         &TPM2_ObjectChangeAuth,
    /* inSize        */         (UINT16)(sizeof(ObjectChangeAuth_In)),
    /* outSize       */         (UINT16)(sizeof(ObjectChangeAuth_Out)),
    /* offsetOfTypes */         offsetof(ObjectChangeAuth_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(ObjectChangeAuth_In,
parentHandle)),
                                 (UINT16)(offsetof(ObjectChangeAuth_In, newAuth))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_AUTH_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_PRIVATE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ObjectChangeAuthDataAddress (&_ObjectChangeAuthData)
#else
#define _ObjectChangeAuthDataAddress 0
#endif // CC_ObjectChangeAuth

#if       CC_CreateLoaded
#include    "CreateLoaded_fp.h"

typedef TPM_RC (CreateLoaded_Entry)(
    CreateLoaded_In*            in,
    CreateLoaded_Out*           out
);


typedef const struct
{
    CreateLoaded_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} CreateLoaded_COMMAND_DESCRIPTOR_t;

CreateLoaded_COMMAND_DESCRIPTOR_t _CreateLoadedData = {
    /* entry         */         &TPM2_CreateLoaded,
    /* inSize        */         (UINT16)(sizeof(CreateLoaded_In)),
    /* outSize       */         (UINT16)(sizeof(CreateLoaded_Out)),
    /* offsetOfTypes */         offsetof(CreateLoaded_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(CreateLoaded_In, inSensitive)),
                                 (UINT16)(offsetof(CreateLoaded_In, inPublic)),
                                 (UINT16)(offsetof(CreateLoaded_Out, outPrivate)),
                                 (UINT16)(offsetof(CreateLoaded_Out, outPublic)),
                                 (UINT16)(offsetof(CreateLoaded_Out, name))},
    /* types         */         {TPMI_DH_PARENT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_SENSITIVE_CREATE_P_UNMARSHAL,
                                 TPM2B_TEMPLATE_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM_HANDLE_H_MARSHAL,
                                 TPM2B_PRIVATE_P_MARSHAL,
                                 TPM2B_PUBLIC_P_MARSHAL,
                                 TPM2B_NAME_P_MARSHAL,
                                 END_OF_LIST}
```

```
};

#define _CreateLoadedDataAddress (&_CreateLoadedData)
#else
#define _CreateLoadedDataAddress 0
#endif // CC_CreateLoaded

#if       CC_Duplicate
#include    "Duplicate_fp.h"

typedef TPM_RC (Duplicate_Entry)(
    Duplicate_In*               in,
    Duplicate_Out*              out
);


typedef const struct
{
    Duplicate_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} Duplicate_COMMAND_DESCRIPTOR_t;

Duplicate_COMMAND_DESCRIPTOR_t _DuplicateData = {
    /* entry         */         &TPM2_Duplicate,
    /* inSize        */         (UINT16)(sizeof(Duplicate_In)),
    /* outSize       */         (UINT16)(sizeof(Duplicate_Out)),
    /* offsetOfTypes */         offsetof(Duplicate_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(Duplicate_In, newParentHandle)),
                                 (UINT16)(offsetof(Duplicate_In, encryptionKeyIn)),
                                 (UINT16)(offsetof(Duplicate_In, symmetricAlg)),
                                 (UINT16)(offsetof(Duplicate_Out, duplicate)),
                                 (UINT16)(offsetof(Duplicate_Out, outSymSeed))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SYM_DEF_OBJECT_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_DATA_P_MARSHAL,
                                 TPM2B_PRIVATE_P_MARSHAL,
                                 TPM2B_ENCRYPTED_SECRET_P_MARSHAL,
                                 END_OF_LIST}
};

#define _DuplicateDataAddress (&_DuplicateData)
#else
#define _DuplicateDataAddress 0
#endif // CC_Duplicate

#if       CC_Rewrap
#include    "Rewrap_fp.h"

typedef TPM_RC (Rewrap_Entry)(
    Rewrap_In*                  in,
    Rewrap_Out*                 out
);


typedef const struct
{
    Rewrap_Entry                *entry;
    UINT16                      inSize;
    UINT16                      outSize;
```

```
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} Rewrap_COMMAND_DESCRIPTOR_t;

Rewrap_COMMAND_DESCRIPTOR_t _RewrapData = {
    /* entry          */        &TPM2_Rewrap,
    /* inSize         */        (UINT16)(sizeof(Rewrap_In)),
    /* outSize        */        (UINT16)(sizeof(Rewrap_Out)),
    /* offsetOfTypes */         offsetof(Rewrap_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */        {(UINT16)(offsetof(Rewrap_In, newParent)),
                                 (UINT16)(offsetof(Rewrap_In, inDuplicate)),
                                 (UINT16)(offsetof(Rewrap_In, name)),
                                 (UINT16)(offsetof(Rewrap_In, inSymSeed)),
                                 (UINT16)(offsetof(Rewrap_Out, outSymSeed))},
    /* types          */        {TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_PRIVATE_P_UNMARSHAL,
                                 TPM2B_NAME_P_UNMARSHAL,
                                 TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_PRIVATE_P_MARSHAL,
                                 TPM2B_ENCRYPTED_SECRET_P_MARSHAL,
                                 END_OF_LIST}
};

#define _RewrapDataAddress (&_RewrapData)
#else
#define _RewrapDataAddress 0
#endif // CC_Rewrap

#if       CC_Import
#include    "Import_fp.h"

typedef TPM_RC (Import_Entry)(
    Import_In*                  in,
    Import_Out*                 out
);


typedef const struct
{
    Import_Entry                *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} Import_COMMAND_DESCRIPTOR_t;

Import_COMMAND_DESCRIPTOR_t _ImportData = {
    /* entry          */        &TPM2_Import,
    /* inSize         */        (UINT16)(sizeof(Import_In)),
    /* outSize        */        (UINT16)(sizeof(Import_Out)),
    /* offsetOfTypes */         offsetof(Import_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */        {(UINT16)(offsetof(Import_In, encryptionKey)),
                                 (UINT16)(offsetof(Import_In, objectPublic)),
                                 (UINT16)(offsetof(Import_In, duplicate)),
                                 (UINT16)(offsetof(Import_In, inSymSeed)),
                                 (UINT16)(offsetof(Import_In, symmetricAlg))},
    /* types          */        {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPM2B_PUBLIC_P_UNMARSHAL,
                                 TPM2B_PRIVATE_P_UNMARSHAL,
                                 TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
                                 TPMT_SYM_DEF_OBJECT_P_UNMARSHAL + ADD_FLAG,
```

```
                                    END_OF_LIST,
                                    TPM2B_PRIVATE_P_MARSHAL,
                                    END_OF_LIST}
};

#define _ImportDataAddress (&_ImportData)
#else
#define _ImportDataAddress 0
#endif // CC_Import

#if       CC_RSA_Encrypt
#include    "RSA_Encrypt_fp.h"

typedef TPM_RC (RSA_Encrypt_Entry)(
    RSA_Encrypt_In*             in,
    RSA_Encrypt_Out*            out
);


typedef const struct
{
    RSA_Encrypt_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} RSA_Encrypt_COMMAND_DESCRIPTOR_t;

RSA_Encrypt_COMMAND_DESCRIPTOR_t _RSA_EncryptData = {
    /* entry         */         &TPM2_RSA_Encrypt,
    /* inSize        */         (UINT16)(sizeof(RSA_Encrypt_In)),
    /* outSize       */         (UINT16)(sizeof(RSA_Encrypt_Out)),
    /* offsetOfTypes */         offsetof(RSA_Encrypt_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(RSA_Encrypt_In, message)),
                                 (UINT16)(offsetof(RSA_Encrypt_In, inScheme)),
                                 (UINT16)(offsetof(RSA_Encrypt_In, label))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL,
                                 TPMT_RSA_DECRYPT_P_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_PUBLIC_KEY_RSA_P_MARSHAL,
                                 END_OF_LIST}
};

#define _RSA_EncryptDataAddress (&_RSA_EncryptData)
#else
#define _RSA_EncryptDataAddress 0
#endif // CC_RSA_Encrypt

#if       CC_RSA_Decrypt
#include    "RSA_Decrypt_fp.h"

typedef TPM_RC (RSA_Decrypt_Entry)(
    RSA_Decrypt_In*             in,
    RSA_Decrypt_Out*            out
);


typedef const struct
{
    RSA_Decrypt_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
```

```
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} RSA_Decrypt_COMMAND_DESCRIPTOR_t;

RSA_Decrypt_COMMAND_DESCRIPTOR_t _RSA_DecryptData = {
    /* entry         */         &TPM2_RSA_Decrypt,
    /* inSize        */         (UINT16)(sizeof(RSA_Decrypt_In)),
    /* outSize       */         (UINT16)(sizeof(RSA_Decrypt_Out)),
    /* offsetOfTypes */         offsetof(RSA_Decrypt_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(RSA_Decrypt_In, cipherText)),
                                 (UINT16)(offsetof(RSA_Decrypt_In, inScheme)),
                                 (UINT16)(offsetof(RSA_Decrypt_In, label))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL,
                                 TPMT_RSA_DECRYPT_P_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_PUBLIC_KEY_RSA_P_MARSHAL,
                                 END_OF_LIST}
};

#define _RSA_DecryptDataAddress (&_RSA_DecryptData)
#else
#define _RSA_DecryptDataAddress 0
#endif // CC_RSA_Decrypt

#if       CC_ECDH_KeyGen
#include    "ECDH_KeyGen_fp.h"

typedef TPM_RC (ECDH_KeyGen_Entry)(
    ECDH_KeyGen_In*             in,
    ECDH_KeyGen_Out*            out
);


typedef const struct
{
    ECDH_KeyGen_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} ECDH_KeyGen_COMMAND_DESCRIPTOR_t;

ECDH_KeyGen_COMMAND_DESCRIPTOR_t _ECDH_KeyGenData = {
    /* entry         */         &TPM2_ECDH_KeyGen,
    /* inSize        */         (UINT16)(sizeof(ECDH_KeyGen_In)),
    /* outSize       */         (UINT16)(sizeof(ECDH_KeyGen_Out)),
    /* offsetOfTypes */         offsetof(ECDH_KeyGen_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ECDH_KeyGen_Out, pubPoint))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ECDH_KeyGenDataAddress (&_ECDH_KeyGenData)
#else
#define _ECDH_KeyGenDataAddress 0
#endif // CC_ECDH_KeyGen

#if       CC_ECDH_ZGen
#include    "ECDH_ZGen_fp.h"
```

```
typedef TPM_RC (ECDH_ZGen_Entry)(
    ECDH_ZGen_In*               in,
    ECDH_ZGen_Out*              out
);


typedef const struct
{
    ECDH_ZGen_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} ECDH_ZGen_COMMAND_DESCRIPTOR_t;

ECDH_ZGen_COMMAND_DESCRIPTOR_t _ECDH_ZGenData = {
    /* entry         */         &TPM2_ECDH_ZGen,
    /* inSize        */         (UINT16)(sizeof(ECDH_ZGen_In)),
    /* outSize       */         (UINT16)(sizeof(ECDH_ZGen_Out)),
    /* offsetOfTypes */         offsetof(ECDH_ZGen_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ECDH_ZGen_In, inPoint))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_ECC_POINT_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ECDH_ZGenDataAddress (&_ECDH_ZGenData)
#else
#define _ECDH_ZGenDataAddress 0
#endif // CC_ECDH_ZGen

#if       CC_ECC_Parameters
#include    "ECC_Parameters_fp.h"

typedef TPM_RC (ECC_Parameters_Entry)(
    ECC_Parameters_In*          in,
    ECC_Parameters_Out*         out
);


typedef const struct
{
    ECC_Parameters_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} ECC_Parameters_COMMAND_DESCRIPTOR_t;

ECC_Parameters_COMMAND_DESCRIPTOR_t _ECC_ParametersData = {
    /* entry         */         &TPM2_ECC_Parameters,
    /* inSize        */         (UINT16)(sizeof(ECC_Parameters_In)),
    /* outSize       */         (UINT16)(sizeof(ECC_Parameters_Out)),
    /* offsetOfTypes */         offsetof(ECC_Parameters_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_ECC_CURVE_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMS_ALGORITHM_DETAIL_ECC_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ECC_ParametersDataAddress (&_ECC_ParametersData)
#else
```

```
#define _ECC_ParametersDataAddress 0
#endif // CC_ECC_Parameters

#if       CC_ZGen_2Phase
#include    "ZGen_2Phase_fp.h"

typedef TPM_RC (ZGen_2Phase_Entry)(
    ZGen_2Phase_In*             in,
    ZGen_2Phase_Out*            out
);


typedef const struct
{
    ZGen_2Phase_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} ZGen_2Phase_COMMAND_DESCRIPTOR_t;

ZGen_2Phase_COMMAND_DESCRIPTOR_t _ZGen_2PhaseData = {
    /* entry         */         &TPM2_ZGen_2Phase,
    /* inSize        */         (UINT16)(sizeof(ZGen_2Phase_In)),
    /* outSize       */         (UINT16)(sizeof(ZGen_2Phase_Out)),
    /* offsetOfTypes */         offsetof(ZGen_2Phase_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ZGen_2Phase_In, inQsB)),
                                 (UINT16)(offsetof(ZGen_2Phase_In, inQeB)),
                                 (UINT16)(offsetof(ZGen_2Phase_In, inScheme)),
                                 (UINT16)(offsetof(ZGen_2Phase_In, counter)),
                                 (UINT16)(offsetof(ZGen_2Phase_Out, outZ2))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_ECC_POINT_P_UNMARSHAL,
                                 TPM2B_ECC_POINT_P_UNMARSHAL,
                                 TPMI_ECC_KEY_EXCHANGE_P_UNMARSHAL,
                                 UINT16_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ZGen_2PhaseDataAddress (&_ZGen_2PhaseData)
#else
#define _ZGen_2PhaseDataAddress 0
#endif // CC_ZGen_2Phase

#if       CC_ECC_Encrypt
#include    "ECC_Encrypt_fp.h"

typedef TPM_RC (ECC_Encrypt_Entry)(
    ECC_Encrypt_In*             in,
    ECC_Encrypt_Out*            out
);


typedef const struct
{
    ECC_Encrypt_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[8];
} ECC_Encrypt_COMMAND_DESCRIPTOR_t;
```

```
ECC_Encrypt_COMMAND_DESCRIPTOR_t _ECC_EncryptData = {
    /* entry         */         &TPM2_ECC_Encrypt,
    /* inSize        */         (UINT16)(sizeof(ECC_Encrypt_In)),
    /* outSize       */         (UINT16)(sizeof(ECC_Encrypt_Out)),
    /* offsetOfTypes */         offsetof(ECC_Encrypt_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ECC_Encrypt_In, plainText)),
                                 (UINT16)(offsetof(ECC_Encrypt_In, inScheme)),
                                 (UINT16)(offsetof(ECC_Encrypt_Out, C2)),
                                 (UINT16)(offsetof(ECC_Encrypt_Out, C3))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 TPMT_KDF_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 TPM2B_MAX_BUFFER_P_MARSHAL,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ECC_EncryptDataAddress (&_ECC_EncryptData)
#else
#define _ECC_EncryptDataAddress 0
#endif // CC_ECC_Encrypt

#if       CC_ECC_Decrypt
#include    "ECC_Decrypt_fp.h"

typedef TPM_RC (ECC_Decrypt_Entry)(
    ECC_Decrypt_In*             in,
    ECC_Decrypt_Out*            out
);


typedef const struct
{
    ECC_Decrypt_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[8];
} ECC_Decrypt_COMMAND_DESCRIPTOR_t;

ECC_Decrypt_COMMAND_DESCRIPTOR_t _ECC_DecryptData = {
    /* entry         */         &TPM2_ECC_Decrypt,
    /* inSize        */         (UINT16)(sizeof(ECC_Decrypt_In)),
    /* outSize       */         (UINT16)(sizeof(ECC_Decrypt_Out)),
    /* offsetOfTypes */         offsetof(ECC_Decrypt_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ECC_Decrypt_In, C1)),
                                 (UINT16)(offsetof(ECC_Decrypt_In, C2)),
                                 (UINT16)(offsetof(ECC_Decrypt_In, C3)),
                                 (UINT16)(offsetof(ECC_Decrypt_In, inScheme))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_ECC_POINT_P_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPMT_KDF_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_MAX_BUFFER_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ECC_DecryptDataAddress (&_ECC_DecryptData)
#else
#define _ECC_DecryptDataAddress 0
```

```
#endif // CC_ECC_Decrypt

#if       CC_EncryptDecrypt
#include    "EncryptDecrypt_fp.h"

typedef TPM_RC (EncryptDecrypt_Entry)(
    EncryptDecrypt_In*          in,
    EncryptDecrypt_Out*         out
);


typedef const struct
{
    EncryptDecrypt_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} EncryptDecrypt_COMMAND_DESCRIPTOR_t;

EncryptDecrypt_COMMAND_DESCRIPTOR_t _EncryptDecryptData = {
    /* entry         */         &TPM2_EncryptDecrypt,
    /* inSize        */         (UINT16)(sizeof(EncryptDecrypt_In)),
    /* outSize       */         (UINT16)(sizeof(EncryptDecrypt_Out)),
    /* offsetOfTypes */         offsetof(EncryptDecrypt_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(EncryptDecrypt_In, decrypt)),
                                 (UINT16)(offsetof(EncryptDecrypt_In, mode)),
                                 (UINT16)(offsetof(EncryptDecrypt_In, ivIn)),
                                 (UINT16)(offsetof(EncryptDecrypt_In, inData)),
                                 (UINT16)(offsetof(EncryptDecrypt_Out, ivOut))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_YES_NO_P_UNMARSHAL,
                                 TPMI_ALG_CIPHER_MODE_P_UNMARSHAL + ADD_FLAG,
                                 TPM2B_IV_P_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_MAX_BUFFER_P_MARSHAL,
                                 TPM2B_IV_P_MARSHAL,
                                 END_OF_LIST}
};

#define _EncryptDecryptDataAddress (&_EncryptDecryptData)
#else
#define _EncryptDecryptDataAddress 0
#endif // CC_EncryptDecrypt

#if       CC_EncryptDecrypt2
#include    "EncryptDecrypt2_fp.h"

typedef TPM_RC (EncryptDecrypt2_Entry)(
    EncryptDecrypt2_In*         in,
    EncryptDecrypt2_Out*        out
);


typedef const struct
{
    EncryptDecrypt2_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} EncryptDecrypt2_COMMAND_DESCRIPTOR_t;
```

```
EncryptDecrypt2_COMMAND_DESCRIPTOR_t _EncryptDecrypt2Data = {
    /* entry          */        &TPM2_EncryptDecrypt2,
    /* inSize         */        (UINT16)(sizeof(EncryptDecrypt2_In)),
    /* outSize        */        (UINT16)(sizeof(EncryptDecrypt2_Out)),
    /* offsetOfTypes */         offsetof(EncryptDecrypt2_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */        {(UINT16)(offsetof(EncryptDecrypt2_In, inData)),
                                 (UINT16)(offsetof(EncryptDecrypt2_In, decrypt)),
                                 (UINT16)(offsetof(EncryptDecrypt2_In, mode)),
                                 (UINT16)(offsetof(EncryptDecrypt2_In, ivIn)),
                                 (UINT16)(offsetof(EncryptDecrypt2_Out, ivOut))},
    /* types          */        {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 TPMI_YES_NO_P_UNMARSHAL,
                                 TPMI_ALG_CIPHER_MODE_P_UNMARSHAL + ADD_FLAG,
                                 TPM2B_IV_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_MAX_BUFFER_P_MARSHAL,
                                 TPM2B_IV_P_MARSHAL,
                                 END_OF_LIST}
};

#define _EncryptDecrypt2DataAddress (&_EncryptDecrypt2Data)
#else
#define _EncryptDecrypt2DataAddress 0
#endif // CC_EncryptDecrypt2

#if       CC_Hash
#include    "Hash_fp.h"

typedef TPM_RC (Hash_Entry)(
    Hash_In*                    in,
    Hash_Out*                   out
);


typedef const struct
{
    Hash_Entry                  *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} Hash_COMMAND_DESCRIPTOR_t;

Hash_COMMAND_DESCRIPTOR_t _HashData = {
    /* entry          */        &TPM2_Hash,
    /* inSize         */        (UINT16)(sizeof(Hash_In)),
    /* outSize        */        (UINT16)(sizeof(Hash_Out)),
    /* offsetOfTypes */         offsetof(Hash_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */        {(UINT16)(offsetof(Hash_In, hashAlg)),
                                 (UINT16)(offsetof(Hash_In, hierarchy)),
                                 (UINT16)(offsetof(Hash_Out, validation))},
    /* types          */        {TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 TPMI_ALG_HASH_P_UNMARSHAL,
                                 TPMI_RH_HIERARCHY_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 TPMT_TK_HASHCHECK_P_MARSHAL,
                                 END_OF_LIST}
};

#define _HashDataAddress (&_HashData)
#else
#define _HashDataAddress 0
#endif // CC_Hash
```

```
#if       CC_HMAC
#include    "HMAC_fp.h"

typedef TPM_RC (HMAC_Entry)(
    HMAC_In*                    in,
    HMAC_Out*                   out
);


typedef const struct
{
    HMAC_Entry                  *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} HMAC_COMMAND_DESCRIPTOR_t;

HMAC_COMMAND_DESCRIPTOR_t _HMACData = {
    /* entry         */         &TPM2_HMAC,
    /* inSize        */         (UINT16)(sizeof(HMAC_In)),
    /* outSize       */         (UINT16)(sizeof(HMAC_Out)),
    /* offsetOfTypes */         offsetof(HMAC_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(HMAC_In, buffer)),
                                 (UINT16)(offsetof(HMAC_In, hashAlg))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 END_OF_LIST}
};

#define _HMACDataAddress (&_HMACData)
#else
#define _HMACDataAddress 0
#endif // CC_HMAC

#if       CC_MAC
#include    "MAC_fp.h"

typedef TPM_RC (MAC_Entry)(
    MAC_In*                     in,
    MAC_Out*                    out
);


typedef const struct
{
    MAC_Entry                   *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} MAC_COMMAND_DESCRIPTOR_t;

MAC_COMMAND_DESCRIPTOR_t _MACData = {
    /* entry         */         &TPM2_MAC,
    /* inSize        */         (UINT16)(sizeof(MAC_In)),
    /* outSize       */         (UINT16)(sizeof(MAC_Out)),
    /* offsetOfTypes */         offsetof(MAC_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(MAC_In, buffer)),
                                 (UINT16)(offsetof(MAC_In, inScheme))},
```

```
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 TPMI_ALG_MAC_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 END_OF_LIST}
};

#define _MACDataAddress (&_MACData)
#else
#define _MACDataAddress 0
#endif // CC_MAC

#if       CC_GetRandom
#include    "GetRandom_fp.h"

typedef TPM_RC (GetRandom_Entry)(
    GetRandom_In*               in,
    GetRandom_Out*              out
);


typedef const struct
{
    GetRandom_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} GetRandom_COMMAND_DESCRIPTOR_t;

GetRandom_COMMAND_DESCRIPTOR_t _GetRandomData = {
    /* entry         */         &TPM2_GetRandom,
    /* inSize        */         (UINT16)(sizeof(GetRandom_In)),
    /* outSize       */         (UINT16)(sizeof(GetRandom_Out)),
    /* offsetOfTypes */         offsetof(GetRandom_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {UINT16_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 END_OF_LIST}
};

#define _GetRandomDataAddress (&_GetRandomData)
#else
#define _GetRandomDataAddress 0
#endif // CC_GetRandom

#if       CC_StirRandom
#include    "StirRandom_fp.h"

typedef TPM_RC (StirRandom_Entry)(
    StirRandom_In*              in
);


typedef const struct
{
    StirRandom_Entry            *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} StirRandom_COMMAND_DESCRIPTOR_t;

StirRandom_COMMAND_DESCRIPTOR_t _StirRandomData = {
```

```
    /* entry          */         &TPM2_StirRandom,
    /* inSize         */         (UINT16)(sizeof(StirRandom_In)),
    /* outSize        */         0,
    /* offsetOfTypes  */         offsetof(StirRandom_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         // No parameter offsets
    /* types          */         {TPM2B_SENSITIVE_DATA_P_UNMARSHAL,
                                  END_OF_LIST,
                                  END_OF_LIST}
};

#define _StirRandomDataAddress (&_StirRandomData)
#else
#define _StirRandomDataAddress 0
#endif // CC_StirRandom

#if       CC_HMAC_Start
#include    "HMAC_Start_fp.h"

typedef TPM_RC (HMAC_Start_Entry)(
    HMAC_Start_In*              in,
    HMAC_Start_Out*             out
);


typedef const struct
{
    HMAC_Start_Entry            *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} HMAC_Start_COMMAND_DESCRIPTOR_t;

HMAC_Start_COMMAND_DESCRIPTOR_t _HMAC_StartData = {
    /* entry          */         &TPM2_HMAC_Start,
    /* inSize         */         (UINT16)(sizeof(HMAC_Start_In)),
    /* outSize        */         (UINT16)(sizeof(HMAC_Start_Out)),
    /* offsetOfTypes  */         offsetof(HMAC_Start_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         {(UINT16)(offsetof(HMAC_Start_In, auth)),
                                  (UINT16)(offsetof(HMAC_Start_In, hashAlg))},
    /* types          */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                  TPM2B_AUTH_P_UNMARSHAL,
                                  TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
                                  END_OF_LIST,
                                  TPMI_DH_OBJECT_H_MARSHAL,
                                  END_OF_LIST}
};

#define _HMAC_StartDataAddress (&_HMAC_StartData)
#else
#define _HMAC_StartDataAddress 0
#endif // CC_HMAC_Start

#if       CC_MAC_Start
#include    "MAC_Start_fp.h"

typedef TPM_RC (MAC_Start_Entry)(
    MAC_Start_In*               in,
    MAC_Start_Out*              out
);


typedef const struct
{
    MAC_Start_Entry             *entry;
```

```
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} MAC_Start_COMMAND_DESCRIPTOR_t;

MAC_Start_COMMAND_DESCRIPTOR_t _MAC_StartData = {
    /* entry         */         &TPM2_MAC_Start,
    /* inSize        */         (UINT16)(sizeof(MAC_Start_In)),
    /* outSize       */         (UINT16)(sizeof(MAC_Start_Out)),
    /* offsetOfTypes */         offsetof(MAC_Start_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(MAC_Start_In, auth)),
                                 (UINT16)(offsetof(MAC_Start_In, inScheme))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_AUTH_P_UNMARSHAL,
                                 TPMI_ALG_MAC_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPMI_DH_OBJECT_H_MARSHAL,
                                 END_OF_LIST}
};

#define _MAC_StartDataAddress (&_MAC_StartData)
#else
#define _MAC_StartDataAddress 0
#endif // CC_MAC_Start

#if       CC_HashSequenceStart
#include    "HashSequenceStart_fp.h"

typedef TPM_RC (HashSequenceStart_Entry)(
    HashSequenceStart_In*       in,
    HashSequenceStart_Out*      out
);


typedef const struct
{
    HashSequenceStart_Entry     *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} HashSequenceStart_COMMAND_DESCRIPTOR_t;

HashSequenceStart_COMMAND_DESCRIPTOR_t _HashSequenceStartData = {
    /* entry         */         &TPM2_HashSequenceStart,
    /* inSize        */         (UINT16)(sizeof(HashSequenceStart_In)),
    /* outSize       */         (UINT16)(sizeof(HashSequenceStart_Out)),
    /* offsetOfTypes */         offsetof(HashSequenceStart_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(HashSequenceStart_In, hashAlg))},
    /* types         */         {TPM2B_AUTH_P_UNMARSHAL,
                                 TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPMI_DH_OBJECT_H_MARSHAL,
                                 END_OF_LIST}
};

#define _HashSequenceStartDataAddress (&_HashSequenceStartData)
#else
#define _HashSequenceStartDataAddress 0
#endif // CC_HashSequenceStart

#if       CC_SequenceUpdate
```

```
#include    "SequenceUpdate_fp.h"

typedef TPM_RC (SequenceUpdate_Entry)(
    SequenceUpdate_In*          in
);


typedef const struct
{
    SequenceUpdate_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} SequenceUpdate_COMMAND_DESCRIPTOR_t;

SequenceUpdate_COMMAND_DESCRIPTOR_t _SequenceUpdateData = {
    /* entry         */         &TPM2_SequenceUpdate,
    /* inSize        */         (UINT16)(sizeof(SequenceUpdate_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(SequenceUpdate_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(SequenceUpdate_In, buffer))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _SequenceUpdateDataAddress (&_SequenceUpdateData)
#else
#define _SequenceUpdateDataAddress 0
#endif // CC_SequenceUpdate

#if       CC_SequenceComplete
#include    "SequenceComplete_fp.h"

typedef TPM_RC (SequenceComplete_Entry)(
    SequenceComplete_In*        in,
    SequenceComplete_Out*       out
);


typedef const struct
{
    SequenceComplete_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} SequenceComplete_COMMAND_DESCRIPTOR_t;

SequenceComplete_COMMAND_DESCRIPTOR_t _SequenceCompleteData = {
    /* entry         */         &TPM2_SequenceComplete,
    /* inSize        */         (UINT16)(sizeof(SequenceComplete_In)),
    /* outSize       */         (UINT16)(sizeof(SequenceComplete_Out)),
    /* offsetOfTypes */         offsetof(SequenceComplete_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(SequenceComplete_In, buffer)),
                                 (UINT16)(offsetof(SequenceComplete_In, hierarchy)),
                                 (UINT16)(offsetof(SequenceComplete_Out,
validation))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 TPMI_RH_HIERARCHY_P_UNMARSHAL + ADD_FLAG,
```

```
                                    END_OF_LIST,
                                    TPM2B_DIGEST_P_MARSHAL,
                                    TPMT_TK_HASHCHECK_P_MARSHAL,
                                    END_OF_LIST}
};

#define _SequenceCompleteDataAddress (&_SequenceCompleteData)
#else
#define _SequenceCompleteDataAddress 0
#endif // CC_SequenceComplete

#if       CC_EventSequenceComplete
#include    "EventSequenceComplete_fp.h"

typedef TPM_RC (EventSequenceComplete_Entry)(
    EventSequenceComplete_In*   in,
    EventSequenceComplete_Out*  out
);


typedef const struct
{
    EventSequenceComplete_Entry *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} EventSequenceComplete_COMMAND_DESCRIPTOR_t;

EventSequenceComplete_COMMAND_DESCRIPTOR_t _EventSequenceCompleteData = {
    /* entry         */         &TPM2_EventSequenceComplete,
    /* inSize        */         (UINT16)(sizeof(EventSequenceComplete_In)),
    /* outSize       */         (UINT16)(sizeof(EventSequenceComplete_Out)),
    /* offsetOfTypes */         offsetof(EventSequenceComplete_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(EventSequenceComplete_In,
sequenceHandle)),
                                 (UINT16)(offsetof(EventSequenceComplete_In,
buffer))},
    /* types         */         {TPMI_DH_PCR_H_UNMARSHAL + ADD_FLAG,
                                 TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPML_DIGEST_VALUES_P_MARSHAL,
                                 END_OF_LIST}
};

#define _EventSequenceCompleteDataAddress (&_EventSequenceCompleteData)
#else
#define _EventSequenceCompleteDataAddress 0
#endif // CC_EventSequenceComplete

#if       CC_Certify
#include    "Certify_fp.h"

typedef TPM_RC (Certify_Entry)(
    Certify_In*                 in,
    Certify_Out*                out
);


typedef const struct
{
    Certify_Entry               *entry;
    UINT16                      inSize;
```

```
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[8];
} Certify_COMMAND_DESCRIPTOR_t;

Certify_COMMAND_DESCRIPTOR_t _CertifyData = {
    /* entry         */         &TPM2_Certify,
    /* inSize        */         (UINT16)(sizeof(Certify_In)),
    /* outSize       */         (UINT16)(sizeof(Certify_Out)),
    /* offsetOfTypes */         offsetof(Certify_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(Certify_In, signHandle)),
                                 (UINT16)(offsetof(Certify_In, qualifyingData)),
                                 (UINT16)(offsetof(Certify_In, inScheme)),
                                 (UINT16)(offsetof(Certify_Out, signature))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_ATTEST_P_MARSHAL,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _CertifyDataAddress (&_CertifyData)
#else
#define _CertifyDataAddress 0
#endif // CC_Certify

#if       CC_CertifyCreation
#include    "CertifyCreation_fp.h"

typedef TPM_RC (CertifyCreation_Entry)(
    CertifyCreation_In*         in,
    CertifyCreation_Out*        out
);


typedef const struct
{
    CertifyCreation_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[6];
    BYTE                        types[10];
} CertifyCreation_COMMAND_DESCRIPTOR_t;

CertifyCreation_COMMAND_DESCRIPTOR_t _CertifyCreationData = {
    /* entry         */         &TPM2_CertifyCreation,
    /* inSize        */         (UINT16)(sizeof(CertifyCreation_In)),
    /* outSize       */         (UINT16)(sizeof(CertifyCreation_Out)),
    /* offsetOfTypes */         offsetof(CertifyCreation_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(CertifyCreation_In, objectHandle)),
                                 (UINT16)(offsetof(CertifyCreation_In,
qualifyingData)),
                                 (UINT16)(offsetof(CertifyCreation_In, creationHash)),
                                 (UINT16)(offsetof(CertifyCreation_In, inScheme)),
                                 (UINT16)(offsetof(CertifyCreation_In,
creationTicket)),
                                 (UINT16)(offsetof(CertifyCreation_Out, signature))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
```

```
                                    TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                    TPMT_TK_CREATION_P_UNMARSHAL,
                                    END_OF_LIST,
                                    TPM2B_ATTEST_P_MARSHAL,
                                    TPMT_SIGNATURE_P_MARSHAL,
                                    END_OF_LIST}
};

#define _CertifyCreationDataAddress (&_CertifyCreationData)
#else
#define _CertifyCreationDataAddress 0
#endif // CC_CertifyCreation

#if       CC_Quote
#include    "Quote_fp.h"

typedef TPM_RC (Quote_Entry)(
    Quote_In*                   in,
    Quote_Out*                  out
);


typedef const struct
{
    Quote_Entry                 *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[8];
} Quote_COMMAND_DESCRIPTOR_t;

Quote_COMMAND_DESCRIPTOR_t _QuoteData = {
    /* entry         */         &TPM2_Quote,
    /* inSize        */         (UINT16)(sizeof(Quote_In)),
    /* outSize       */         (UINT16)(sizeof(Quote_Out)),
    /* offsetOfTypes */         offsetof(Quote_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(Quote_In, qualifyingData)),
                                 (UINT16)(offsetof(Quote_In, inScheme)),
                                 (UINT16)(offsetof(Quote_In, PCRselect)),
                                 (UINT16)(offsetof(Quote_Out, signature))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 TPML_PCR_SELECTION_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ATTEST_P_MARSHAL,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _QuoteDataAddress (&_QuoteData)
#else
#define _QuoteDataAddress 0
#endif // CC_Quote

#if       CC_GetSessionAuditDigest
#include    "GetSessionAuditDigest_fp.h"

typedef TPM_RC (GetSessionAuditDigest_Entry)(
    GetSessionAuditDigest_In*   in,
    GetSessionAuditDigest_Out*  out
);


typedef const struct
```

```
{
    GetSessionAuditDigest_Entry *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[9];
} GetSessionAuditDigest_COMMAND_DESCRIPTOR_t;

GetSessionAuditDigest_COMMAND_DESCRIPTOR_t _GetSessionAuditDigestData = {
    /* entry         */         &TPM2_GetSessionAuditDigest,
    /* inSize        */         (UINT16)(sizeof(GetSessionAuditDigest_In)),
    /* outSize       */         (UINT16)(sizeof(GetSessionAuditDigest_Out)),
    /* offsetOfTypes */         offsetof(GetSessionAuditDigest_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(GetSessionAuditDigest_In,
signHandle)),
                                 (UINT16)(offsetof(GetSessionAuditDigest_In,
sessionHandle)),
                                 (UINT16)(offsetof(GetSessionAuditDigest_In,
qualifyingData)),
                                 (UINT16)(offsetof(GetSessionAuditDigest_In,
inScheme)),
                                 (UINT16)(offsetof(GetSessionAuditDigest_Out,
signature))},
    /* types         */         {TPMI_RH_ENDORSEMENT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPMI_SH_HMAC_H_UNMARSHAL,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_ATTEST_P_MARSHAL,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _GetSessionAuditDigestDataAddress (&_GetSessionAuditDigestData)
#else
#define _GetSessionAuditDigestDataAddress 0
#endif // CC_GetSessionAuditDigest

#if       CC_GetCommandAuditDigest
#include    "GetCommandAuditDigest_fp.h"

typedef TPM_RC (GetCommandAuditDigest_Entry)(
    GetCommandAuditDigest_In*   in,
    GetCommandAuditDigest_Out*  out
);


typedef const struct
{
    GetCommandAuditDigest_Entry *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[8];
} GetCommandAuditDigest_COMMAND_DESCRIPTOR_t;

GetCommandAuditDigest_COMMAND_DESCRIPTOR_t _GetCommandAuditDigestData = {
    /* entry         */         &TPM2_GetCommandAuditDigest,
    /* inSize        */         (UINT16)(sizeof(GetCommandAuditDigest_In)),
    /* outSize       */         (UINT16)(sizeof(GetCommandAuditDigest_Out)),
    /* offsetOfTypes */         offsetof(GetCommandAuditDigest_COMMAND_DESCRIPTOR_t,
types),
```

```
    /* offsets       */         {(UINT16)(offsetof(GetCommandAuditDigest_In,
signHandle)),
                                 (UINT16)(offsetof(GetCommandAuditDigest_In,
qualifyingData)),
                                 (UINT16)(offsetof(GetCommandAuditDigest_In,
inScheme)),
                                 (UINT16)(offsetof(GetCommandAuditDigest_Out,
signature))},
    /* types         */         {TPMI_RH_ENDORSEMENT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_ATTEST_P_MARSHAL,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _GetCommandAuditDigestDataAddress (&_GetCommandAuditDigestData)
#else
#define _GetCommandAuditDigestDataAddress 0
#endif // CC_GetCommandAuditDigest

#if       CC_GetTime
#include    "GetTime_fp.h"

typedef TPM_RC (GetTime_Entry)(
    GetTime_In*                 in,
    GetTime_Out*                out
);


typedef const struct
{
    GetTime_Entry               *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[8];
} GetTime_COMMAND_DESCRIPTOR_t;

GetTime_COMMAND_DESCRIPTOR_t _GetTimeData = {
    /* entry         */         &TPM2_GetTime,
    /* inSize        */         (UINT16)(sizeof(GetTime_In)),
    /* outSize       */         (UINT16)(sizeof(GetTime_Out)),
    /* offsetOfTypes */         offsetof(GetTime_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(GetTime_In, signHandle)),
                                 (UINT16)(offsetof(GetTime_In, qualifyingData)),
                                 (UINT16)(offsetof(GetTime_In, inScheme)),
                                 (UINT16)(offsetof(GetTime_Out, signature))},
    /* types         */         {TPMI_RH_ENDORSEMENT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 TPM2B_ATTEST_P_MARSHAL,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _GetTimeDataAddress (&_GetTimeData)
#else
#define _GetTimeDataAddress 0
#endif // CC_GetTime
```

```
#if       CC_CertifyX509
#include    "CertifyX509_fp.h"

typedef TPM_RC (CertifyX509_Entry)(
    CertifyX509_In*             in,
    CertifyX509_Out*            out
);


typedef const struct
{
    CertifyX509_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[6];
    BYTE                        types[10];
} CertifyX509_COMMAND_DESCRIPTOR_t;

CertifyX509_COMMAND_DESCRIPTOR_t _CertifyX509Data = {
    /* entry         */         &TPM2_CertifyX509,
    /* inSize        */         (UINT16)(sizeof(CertifyX509_In)),
    /* outSize       */         (UINT16)(sizeof(CertifyX509_Out)),
    /* offsetOfTypes */         offsetof(CertifyX509_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(CertifyX509_In, signHandle)),
                                 (UINT16)(offsetof(CertifyX509_In, reserved)),
                                 (UINT16)(offsetof(CertifyX509_In, inScheme)),
                                 (UINT16)(offsetof(CertifyX509_In,
partialCertificate)),
                                 (UINT16)(offsetof(CertifyX509_Out, tbsDigest)),
                                 (UINT16)(offsetof(CertifyX509_Out, signature))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_MAX_BUFFER_P_MARSHAL,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _CertifyX509DataAddress (&_CertifyX509Data)
#else
#define _CertifyX509DataAddress 0
#endif // CC_CertifyX509

#if       CC_Commit
#include    "Commit_fp.h"

typedef TPM_RC (Commit_Entry)(
    Commit_In*                  in,
    Commit_Out*                 out
);


typedef const struct
{
    Commit_Entry                *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[6];
    BYTE                        types[10];
} Commit_COMMAND_DESCRIPTOR_t;
```

```
Commit_COMMAND_DESCRIPTOR_t _CommitData = {
    /* entry         */         &TPM2_Commit,
    /* inSize        */         (UINT16)(sizeof(Commit_In)),
    /* outSize       */         (UINT16)(sizeof(Commit_Out)),
    /* offsetOfTypes */         offsetof(Commit_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(Commit_In, P1)),
                                 (UINT16)(offsetof(Commit_In, s2)),
                                 (UINT16)(offsetof(Commit_In, y2)),
                                 (UINT16)(offsetof(Commit_Out, L)),
                                 (UINT16)(offsetof(Commit_Out, E)),
                                 (UINT16)(offsetof(Commit_Out, counter))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_ECC_POINT_P_UNMARSHAL,
                                 TPM2B_SENSITIVE_DATA_P_UNMARSHAL,
                                 TPM2B_ECC_PARAMETER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 UINT16_P_MARSHAL,
                                 END_OF_LIST}
};

#define _CommitDataAddress (&_CommitData)
#else
#define _CommitDataAddress 0
#endif // CC_Commit

#if       CC_EC_Ephemeral
#include    "EC_Ephemeral_fp.h"

typedef TPM_RC (EC_Ephemeral_Entry)(
    EC_Ephemeral_In*            in,
    EC_Ephemeral_Out*           out
);


typedef const struct
{
    EC_Ephemeral_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} EC_Ephemeral_COMMAND_DESCRIPTOR_t;

EC_Ephemeral_COMMAND_DESCRIPTOR_t _EC_EphemeralData = {
    /* entry         */         &TPM2_EC_Ephemeral,
    /* inSize        */         (UINT16)(sizeof(EC_Ephemeral_In)),
    /* outSize       */         (UINT16)(sizeof(EC_Ephemeral_Out)),
    /* offsetOfTypes */         offsetof(EC_Ephemeral_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(EC_Ephemeral_Out, counter))},
    /* types         */         {TPMI_ECC_CURVE_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ECC_POINT_P_MARSHAL,
                                 UINT16_P_MARSHAL,
                                 END_OF_LIST}
};

#define _EC_EphemeralDataAddress (&_EC_EphemeralData)
#else
#define _EC_EphemeralDataAddress 0
#endif // CC_EC_Ephemeral
```

```
#if       CC_VerifySignature
#include    "VerifySignature_fp.h"

typedef TPM_RC (VerifySignature_Entry)(
    VerifySignature_In*         in,
    VerifySignature_Out*        out
);


typedef const struct
{
    VerifySignature_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} VerifySignature_COMMAND_DESCRIPTOR_t;

VerifySignature_COMMAND_DESCRIPTOR_t _VerifySignatureData = {
    /* entry         */         &TPM2_VerifySignature,
    /* inSize        */         (UINT16)(sizeof(VerifySignature_In)),
    /* outSize       */         (UINT16)(sizeof(VerifySignature_Out)),
    /* offsetOfTypes */         offsetof(VerifySignature_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(VerifySignature_In, digest)),
                                 (UINT16)(offsetof(VerifySignature_In, signature))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPMT_SIGNATURE_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMT_TK_VERIFIED_P_MARSHAL,
                                 END_OF_LIST}
};

#define _VerifySignatureDataAddress (&_VerifySignatureData)
#else
#define _VerifySignatureDataAddress 0
#endif // CC_VerifySignature

#if       CC_Sign
#include    "Sign_fp.h"

typedef TPM_RC (Sign_Entry)(
    Sign_In*                    in,
    Sign_Out*                   out
);


typedef const struct
{
    Sign_Entry                  *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} Sign_COMMAND_DESCRIPTOR_t;

Sign_COMMAND_DESCRIPTOR_t _SignData = {
    /* entry         */         &TPM2_Sign,
    /* inSize        */         (UINT16)(sizeof(Sign_In)),
    /* outSize       */         (UINT16)(sizeof(Sign_Out)),
    /* offsetOfTypes */         offsetof(Sign_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(Sign_In, digest)),
                                 (UINT16)(offsetof(Sign_In, inScheme)),
                                 (UINT16)(offsetof(Sign_In, validation))},
```

```
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 TPMT_TK_HASHCHECK_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _SignDataAddress (&_SignData)
#else
#define _SignDataAddress 0
#endif // CC_Sign

#if       CC_SetCommandCodeAuditStatus
#include    "SetCommandCodeAuditStatus_fp.h"

typedef TPM_RC (SetCommandCodeAuditStatus_Entry)(
    SetCommandCodeAuditStatus_In* in
);


typedef const struct
{
    SetCommandCodeAuditStatus_Entry *entry;
    UINT16                          inSize;
    UINT16                          outSize;
    UINT16                          offsetOfTypes;
    UINT16                          paramOffsets[3];
    BYTE                            types[6];
} SetCommandCodeAuditStatus_COMMAND_DESCRIPTOR_t;

SetCommandCodeAuditStatus_COMMAND_DESCRIPTOR_t _SetCommandCodeAuditStatusData = {
    /* entry         */         &TPM2_SetCommandCodeAuditStatus,
    /* inSize        */         (UINT16)(sizeof(SetCommandCodeAuditStatus_In)),
    /* outSize       */         0,
    /* offsetOfTypes */
offsetof(SetCommandCodeAuditStatus_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(SetCommandCodeAuditStatus_In,
auditAlg)),
                                 (UINT16)(offsetof(SetCommandCodeAuditStatus_In,
setList)),
                                 (UINT16)(offsetof(SetCommandCodeAuditStatus_In,
clearList))},
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
                                 TPML_CC_P_UNMARSHAL,
                                 TPML_CC_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _SetCommandCodeAuditStatusDataAddress (&_SetCommandCodeAuditStatusData)
#else
#define _SetCommandCodeAuditStatusDataAddress 0
#endif // CC_SetCommandCodeAuditStatus

#if       CC_PCR_Extend
#include    "PCR_Extend_fp.h"

typedef TPM_RC (PCR_Extend_Entry)(
    PCR_Extend_In*              in
);


typedef const struct
```

```
{
    PCR_Extend_Entry            *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PCR_Extend_COMMAND_DESCRIPTOR_t;

PCR_Extend_COMMAND_DESCRIPTOR_t _PCR_ExtendData = {
    /* entry         */         &TPM2_PCR_Extend,
    /* inSize        */         (UINT16)(sizeof(PCR_Extend_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PCR_Extend_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PCR_Extend_In, digests))},
    /* types         */         {TPMI_DH_PCR_H_UNMARSHAL + ADD_FLAG,
                                 TPML_DIGEST_VALUES_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PCR_ExtendDataAddress (&_PCR_ExtendData)
#else
#define _PCR_ExtendDataAddress 0
#endif // CC_PCR_Extend

#if       CC_PCR_Event
#include    "PCR_Event_fp.h"

typedef TPM_RC (PCR_Event_Entry)(
    PCR_Event_In*               in,
    PCR_Event_Out*              out
);


typedef const struct
{
    PCR_Event_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} PCR_Event_COMMAND_DESCRIPTOR_t;

PCR_Event_COMMAND_DESCRIPTOR_t _PCR_EventData = {
    /* entry         */         &TPM2_PCR_Event,
    /* inSize        */         (UINT16)(sizeof(PCR_Event_In)),
    /* outSize       */         (UINT16)(sizeof(PCR_Event_Out)),
    /* offsetOfTypes */         offsetof(PCR_Event_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PCR_Event_In, eventData))},
    /* types         */         {TPMI_DH_PCR_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_EVENT_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPML_DIGEST_VALUES_P_MARSHAL,
                                 END_OF_LIST}
};

#define _PCR_EventDataAddress (&_PCR_EventData)
#else
#define _PCR_EventDataAddress 0
#endif // CC_PCR_Event

#if       CC_PCR_Read
#include    "PCR_Read_fp.h"
```

```
typedef TPM_RC (PCR_Read_Entry)(
    PCR_Read_In*                in,
    PCR_Read_Out*               out
);


typedef const struct
{
    PCR_Read_Entry              *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[6];
} PCR_Read_COMMAND_DESCRIPTOR_t;

PCR_Read_COMMAND_DESCRIPTOR_t _PCR_ReadData = {
    /* entry         */         &TPM2_PCR_Read,
    /* inSize        */         (UINT16)(sizeof(PCR_Read_In)),
    /* outSize       */         (UINT16)(sizeof(PCR_Read_Out)),
    /* offsetOfTypes */         offsetof(PCR_Read_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PCR_Read_Out, pcrSelectionOut)),
                                 (UINT16)(offsetof(PCR_Read_Out, pcrValues))},
    /* types         */         {TPML_PCR_SELECTION_P_UNMARSHAL,
                                 END_OF_LIST,
                                 UINT32_P_MARSHAL,
                                 TPML_PCR_SELECTION_P_MARSHAL,
                                 TPML_DIGEST_P_MARSHAL,
                                 END_OF_LIST}
};

#define _PCR_ReadDataAddress (&_PCR_ReadData)
#else
#define _PCR_ReadDataAddress 0
#endif // CC_PCR_Read

#if       CC_PCR_Allocate
#include    "PCR_Allocate_fp.h"

typedef TPM_RC (PCR_Allocate_Entry)(
    PCR_Allocate_In*            in,
    PCR_Allocate_Out*           out
);


typedef const struct
{
    PCR_Allocate_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[8];
} PCR_Allocate_COMMAND_DESCRIPTOR_t;

PCR_Allocate_COMMAND_DESCRIPTOR_t _PCR_AllocateData = {
    /* entry         */         &TPM2_PCR_Allocate,
    /* inSize        */         (UINT16)(sizeof(PCR_Allocate_In)),
    /* outSize       */         (UINT16)(sizeof(PCR_Allocate_Out)),
    /* offsetOfTypes */         offsetof(PCR_Allocate_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PCR_Allocate_In, pcrAllocation)),
                                 (UINT16)(offsetof(PCR_Allocate_Out, maxPCR)),
                                 (UINT16)(offsetof(PCR_Allocate_Out, sizeNeeded)),
                                 (UINT16)(offsetof(PCR_Allocate_Out, sizeAvailable))},
    /* types         */         {TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 TPML_PCR_SELECTION_P_UNMARSHAL,
```

```
                                    END_OF_LIST,
                                    TPMI_YES_NO_P_MARSHAL,
                                    UINT32_P_MARSHAL,
                                    UINT32_P_MARSHAL,
                                    UINT32_P_MARSHAL,
                                    END_OF_LIST}
};

#define _PCR_AllocateDataAddress (&_PCR_AllocateData)
#else
#define _PCR_AllocateDataAddress 0
#endif // CC_PCR_Allocate

#if       CC_PCR_SetAuthPolicy
#include    "PCR_SetAuthPolicy_fp.h"

typedef TPM_RC (PCR_SetAuthPolicy_Entry)(
    PCR_SetAuthPolicy_In*       in
);


typedef const struct
{
    PCR_SetAuthPolicy_Entry     *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[6];
} PCR_SetAuthPolicy_COMMAND_DESCRIPTOR_t;

PCR_SetAuthPolicy_COMMAND_DESCRIPTOR_t _PCR_SetAuthPolicyData = {
    /* entry         */         &TPM2_PCR_SetAuthPolicy,
    /* inSize        */         (UINT16)(sizeof(PCR_SetAuthPolicy_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PCR_SetAuthPolicy_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PCR_SetAuthPolicy_In, authPolicy)),
                                 (UINT16)(offsetof(PCR_SetAuthPolicy_In, hashAlg)),
                                 (UINT16)(offsetof(PCR_SetAuthPolicy_In, pcrNum))},
    /* types         */         {TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
                                 TPMI_DH_PCR_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PCR_SetAuthPolicyDataAddress (&_PCR_SetAuthPolicyData)
#else
#define _PCR_SetAuthPolicyDataAddress 0
#endif // CC_PCR_SetAuthPolicy

#if       CC_PCR_SetAuthValue
#include    "PCR_SetAuthValue_fp.h"

typedef TPM_RC (PCR_SetAuthValue_Entry)(
    PCR_SetAuthValue_In*        in
);


typedef const struct
{
    PCR_SetAuthValue_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
```

```
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PCR_SetAuthValue_COMMAND_DESCRIPTOR_t;

PCR_SetAuthValue_COMMAND_DESCRIPTOR_t _PCR_SetAuthValueData = {
    /* entry         */         &TPM2_PCR_SetAuthValue,
    /* inSize        */         (UINT16)(sizeof(PCR_SetAuthValue_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PCR_SetAuthValue_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PCR_SetAuthValue_In, auth))},
    /* types         */         {TPMI_DH_PCR_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PCR_SetAuthValueDataAddress (&_PCR_SetAuthValueData)
#else
#define _PCR_SetAuthValueDataAddress 0
#endif // CC_PCR_SetAuthValue

#if       CC_PCR_Reset
#include    "PCR_Reset_fp.h"

typedef TPM_RC (PCR_Reset_Entry)(
    PCR_Reset_In*               in
);


typedef const struct
{
    PCR_Reset_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} PCR_Reset_COMMAND_DESCRIPTOR_t;

PCR_Reset_COMMAND_DESCRIPTOR_t _PCR_ResetData = {
    /* entry         */         &TPM2_PCR_Reset,
    /* inSize        */         (UINT16)(sizeof(PCR_Reset_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PCR_Reset_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_DH_PCR_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PCR_ResetDataAddress (&_PCR_ResetData)
#else
#define _PCR_ResetDataAddress 0
#endif // CC_PCR_Reset

#if       CC_PolicySigned
#include    "PolicySigned_fp.h"

typedef TPM_RC (PolicySigned_Entry)(
    PolicySigned_In*            in,
    PolicySigned_Out*           out
);


typedef const struct
```

```
{
    PolicySigned_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[7];
    BYTE                        types[11];
} PolicySigned_COMMAND_DESCRIPTOR_t;

PolicySigned_COMMAND_DESCRIPTOR_t _PolicySignedData = {
    /* entry         */         &TPM2_PolicySigned,
    /* inSize        */         (UINT16)(sizeof(PolicySigned_In)),
    /* outSize       */         (UINT16)(sizeof(PolicySigned_Out)),
    /* offsetOfTypes */         offsetof(PolicySigned_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicySigned_In, policySession)),
                                 (UINT16)(offsetof(PolicySigned_In, nonceTPM)),
                                 (UINT16)(offsetof(PolicySigned_In, cpHashA)),
                                 (UINT16)(offsetof(PolicySigned_In, policyRef)),
                                 (UINT16)(offsetof(PolicySigned_In, expiration)),
                                 (UINT16)(offsetof(PolicySigned_In, auth)),
                                 (UINT16)(offsetof(PolicySigned_Out, policyTicket))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_NONCE_P_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPM2B_NONCE_P_UNMARSHAL,
                                 INT32_P_UNMARSHAL,
                                 TPMT_SIGNATURE_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_TIMEOUT_P_MARSHAL,
                                 TPMT_TK_AUTH_P_MARSHAL,
                                 END_OF_LIST}
};

#define _PolicySignedDataAddress (&_PolicySignedData)
#else
#define _PolicySignedDataAddress 0
#endif // CC_PolicySigned

#if       CC_PolicySecret
#include    "PolicySecret_fp.h"

typedef TPM_RC (PolicySecret_Entry)(
    PolicySecret_In*            in,
    PolicySecret_Out*           out
);


typedef const struct
{
    PolicySecret_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[6];
    BYTE                        types[10];
} PolicySecret_COMMAND_DESCRIPTOR_t;

PolicySecret_COMMAND_DESCRIPTOR_t _PolicySecretData = {
    /* entry         */         &TPM2_PolicySecret,
    /* inSize        */         (UINT16)(sizeof(PolicySecret_In)),
    /* outSize       */         (UINT16)(sizeof(PolicySecret_Out)),
    /* offsetOfTypes */         offsetof(PolicySecret_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicySecret_In, policySession)),
                                 (UINT16)(offsetof(PolicySecret_In, nonceTPM)),
                                 (UINT16)(offsetof(PolicySecret_In, cpHashA)),
```

```
                                    (UINT16)(offsetof(PolicySecret_In, policyRef)),
                                    (UINT16)(offsetof(PolicySecret_In, expiration)),
                                    (UINT16)(offsetof(PolicySecret_Out, policyTicket))},
    /* types         */             {TPMI_DH_ENTITY_H_UNMARSHAL,
                                    TPMI_SH_POLICY_H_UNMARSHAL,
                                    TPM2B_NONCE_P_UNMARSHAL,
                                    TPM2B_DIGEST_P_UNMARSHAL,
                                    TPM2B_NONCE_P_UNMARSHAL,
                                    INT32_P_UNMARSHAL,
                                    END_OF_LIST,
                                    TPM2B_TIMEOUT_P_MARSHAL,
                                    TPMT_TK_AUTH_P_MARSHAL,
                                    END_OF_LIST}
};

#define _PolicySecretDataAddress (&_PolicySecretData)
#else
#define _PolicySecretDataAddress 0
#endif // CC_PolicySecret

#if       CC_PolicyTicket
#include    "PolicyTicket_fp.h"

typedef TPM_RC (PolicyTicket_Entry)(
    PolicyTicket_In*            in
);


typedef const struct
{
    PolicyTicket_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[8];
} PolicyTicket_COMMAND_DESCRIPTOR_t;

PolicyTicket_COMMAND_DESCRIPTOR_t _PolicyTicketData = {
    /* entry         */             &TPM2_PolicyTicket,
    /* inSize        */             (UINT16)(sizeof(PolicyTicket_In)),
    /* outSize       */             0,
    /* offsetOfTypes */             offsetof(PolicyTicket_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */             {(UINT16)(offsetof(PolicyTicket_In, timeout)),
                                    (UINT16)(offsetof(PolicyTicket_In, cpHashA)),
                                    (UINT16)(offsetof(PolicyTicket_In, policyRef)),
                                    (UINT16)(offsetof(PolicyTicket_In, authName)),
                                    (UINT16)(offsetof(PolicyTicket_In, ticket))},
    /* types         */             {TPMI_SH_POLICY_H_UNMARSHAL,
                                    TPM2B_TIMEOUT_P_UNMARSHAL,
                                    TPM2B_DIGEST_P_UNMARSHAL,
                                    TPM2B_NONCE_P_UNMARSHAL,
                                    TPM2B_NAME_P_UNMARSHAL,
                                    TPMT_TK_AUTH_P_UNMARSHAL,
                                    END_OF_LIST,
                                    END_OF_LIST}
};

#define _PolicyTicketDataAddress (&_PolicyTicketData)
#else
#define _PolicyTicketDataAddress 0
#endif // CC_PolicyTicket

#if       CC_PolicyOR
#include    "PolicyOR_fp.h"
```

```
typedef TPM_RC (PolicyOR_Entry)(
    PolicyOR_In*                in
);


typedef const struct
{
    PolicyOR_Entry              *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyOR_COMMAND_DESCRIPTOR_t;

PolicyOR_COMMAND_DESCRIPTOR_t _PolicyORData = {
    /* entry         */         &TPM2_PolicyOR,
    /* inSize        */         (UINT16)(sizeof(PolicyOR_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyOR_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyOR_In, pHashList))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPML_DIGEST_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyORDataAddress (&_PolicyORData)
#else
#define _PolicyORDataAddress 0
#endif // CC_PolicyOR

#if       CC_PolicyPCR
#include    "PolicyPCR_fp.h"

typedef TPM_RC (PolicyPCR_Entry)(
    PolicyPCR_In*               in
);


typedef const struct
{
    PolicyPCR_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} PolicyPCR_COMMAND_DESCRIPTOR_t;

PolicyPCR_COMMAND_DESCRIPTOR_t _PolicyPCRData = {
    /* entry         */         &TPM2_PolicyPCR,
    /* inSize        */         (UINT16)(sizeof(PolicyPCR_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyPCR_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyPCR_In, pcrDigest)),
                                 (UINT16)(offsetof(PolicyPCR_In, pcrs))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPML_PCR_SELECTION_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyPCRDataAddress (&_PolicyPCRData)
#else
```

```
#define _PolicyPCRDataAddress 0
#endif // CC_PolicyPCR

#if       CC_PolicyLocality
#include    "PolicyLocality_fp.h"

typedef TPM_RC (PolicyLocality_Entry)(
    PolicyLocality_In*          in
);


typedef const struct
{
    PolicyLocality_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyLocality_COMMAND_DESCRIPTOR_t;

PolicyLocality_COMMAND_DESCRIPTOR_t _PolicyLocalityData = {
    /* entry         */         &TPM2_PolicyLocality,
    /* inSize        */         (UINT16)(sizeof(PolicyLocality_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyLocality_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyLocality_In, locality))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPMA_LOCALITY_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyLocalityDataAddress (&_PolicyLocalityData)
#else
#define _PolicyLocalityDataAddress 0
#endif // CC_PolicyLocality

#if       CC_PolicyNV
#include    "PolicyNV_fp.h"

typedef TPM_RC (PolicyNV_Entry)(
    PolicyNV_In*                in
);


typedef const struct
{
    PolicyNV_Entry              *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[8];
} PolicyNV_COMMAND_DESCRIPTOR_t;

PolicyNV_COMMAND_DESCRIPTOR_t _PolicyNVData = {
    /* entry         */         &TPM2_PolicyNV,
    /* inSize        */         (UINT16)(sizeof(PolicyNV_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyNV_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyNV_In, nvIndex)),
                                 (UINT16)(offsetof(PolicyNV_In, policySession)),
                                 (UINT16)(offsetof(PolicyNV_In, operandB)),
                                 (UINT16)(offsetof(PolicyNV_In, offset)),
                                 (UINT16)(offsetof(PolicyNV_In, operation))},
```

```
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_OPERAND_P_UNMARSHAL,
                                 UINT16_P_UNMARSHAL,
                                 TPM_EO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyNVDataAddress (&_PolicyNVData)
#else
#define _PolicyNVDataAddress 0
#endif // CC_PolicyNV

#if       CC_PolicyCounterTimer
#include    "PolicyCounterTimer_fp.h"

typedef TPM_RC (PolicyCounterTimer_Entry)(
    PolicyCounterTimer_In*      in
);


typedef const struct
{
    PolicyCounterTimer_Entry    *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[6];
} PolicyCounterTimer_COMMAND_DESCRIPTOR_t;

PolicyCounterTimer_COMMAND_DESCRIPTOR_t _PolicyCounterTimerData = {
    /* entry         */         &TPM2_PolicyCounterTimer,
    /* inSize        */         (UINT16)(sizeof(PolicyCounterTimer_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyCounterTimer_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PolicyCounterTimer_In, operandB)),
                                 (UINT16)(offsetof(PolicyCounterTimer_In, offset)),
                                 (UINT16)(offsetof(PolicyCounterTimer_In,
operation))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_OPERAND_P_UNMARSHAL,
                                 UINT16_P_UNMARSHAL,
                                 TPM_EO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyCounterTimerDataAddress (&_PolicyCounterTimerData)
#else
#define _PolicyCounterTimerDataAddress 0
#endif // CC_PolicyCounterTimer

#if       CC_PolicyCommandCode
#include    "PolicyCommandCode_fp.h"

typedef TPM_RC (PolicyCommandCode_Entry)(
    PolicyCommandCode_In*       in
);


typedef const struct
{
```

```
    PolicyCommandCode_Entry     *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyCommandCode_COMMAND_DESCRIPTOR_t;

PolicyCommandCode_COMMAND_DESCRIPTOR_t _PolicyCommandCodeData = {
    /* entry         */         &TPM2_PolicyCommandCode,
    /* inSize        */         (UINT16)(sizeof(PolicyCommandCode_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyCommandCode_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PolicyCommandCode_In, code))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM_CC_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyCommandCodeDataAddress (&_PolicyCommandCodeData)
#else
#define _PolicyCommandCodeDataAddress 0
#endif // CC_PolicyCommandCode

#if       CC_PolicyPhysicalPresence
#include    "PolicyPhysicalPresence_fp.h"

typedef TPM_RC (PolicyPhysicalPresence_Entry)(
    PolicyPhysicalPresence_In*  in
);


typedef const struct
{
    PolicyPhysicalPresence_Entry    *entry;
    UINT16                          inSize;
    UINT16                          outSize;
    UINT16                          offsetOfTypes;
    BYTE                            types[3];
} PolicyPhysicalPresence_COMMAND_DESCRIPTOR_t;

PolicyPhysicalPresence_COMMAND_DESCRIPTOR_t _PolicyPhysicalPresenceData = {
    /* entry         */         &TPM2_PolicyPhysicalPresence,
    /* inSize        */         (UINT16)(sizeof(PolicyPhysicalPresence_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyPhysicalPresence_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyPhysicalPresenceDataAddress (&_PolicyPhysicalPresenceData)
#else
#define _PolicyPhysicalPresenceDataAddress 0
#endif // CC_PolicyPhysicalPresence

#if       CC_PolicyCpHash
#include    "PolicyCpHash_fp.h"

typedef TPM_RC (PolicyCpHash_Entry)(
    PolicyCpHash_In*            in
);
```

```
typedef const struct
{
    PolicyCpHash_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyCpHash_COMMAND_DESCRIPTOR_t;

PolicyCpHash_COMMAND_DESCRIPTOR_t _PolicyCpHashData = {
    /* entry         */         &TPM2_PolicyCpHash,
    /* inSize        */         (UINT16)(sizeof(PolicyCpHash_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyCpHash_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyCpHash_In, cpHashA))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyCpHashDataAddress (&_PolicyCpHashData)
#else
#define _PolicyCpHashDataAddress 0
#endif // CC_PolicyCpHash

#if       CC_PolicyNameHash
#include    "PolicyNameHash_fp.h"

typedef TPM_RC (PolicyNameHash_Entry)(
    PolicyNameHash_In*          in
);


typedef const struct
{
    PolicyNameHash_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyNameHash_COMMAND_DESCRIPTOR_t;

PolicyNameHash_COMMAND_DESCRIPTOR_t _PolicyNameHashData = {
    /* entry         */         &TPM2_PolicyNameHash,
    /* inSize        */         (UINT16)(sizeof(PolicyNameHash_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyNameHash_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyNameHash_In, nameHash))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyNameHashDataAddress (&_PolicyNameHashData)
#else
#define _PolicyNameHashDataAddress 0
#endif // CC_PolicyNameHash

#if       CC_PolicyDuplicationSelect
#include    "PolicyDuplicationSelect_fp.h"
```

```
typedef TPM_RC (PolicyDuplicationSelect_Entry)(
    PolicyDuplicationSelect_In* in
);


typedef const struct
{
    PolicyDuplicationSelect_Entry   *entry;
    UINT16                          inSize;
    UINT16                          outSize;
    UINT16                          offsetOfTypes;
    UINT16                          paramOffsets[3];
    BYTE                            types[6];
} PolicyDuplicationSelect_COMMAND_DESCRIPTOR_t;

PolicyDuplicationSelect_COMMAND_DESCRIPTOR_t _PolicyDuplicationSelectData = {
    /* entry         */         &TPM2_PolicyDuplicationSelect,
    /* inSize        */         (UINT16)(sizeof(PolicyDuplicationSelect_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyDuplicationSelect_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PolicyDuplicationSelect_In,
objectName)),
                                 (UINT16)(offsetof(PolicyDuplicationSelect_In,
newParentName)),
                                 (UINT16)(offsetof(PolicyDuplicationSelect_In,
includeObject))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_NAME_P_UNMARSHAL,
                                 TPM2B_NAME_P_UNMARSHAL,
                                 TPMI_YES_NO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyDuplicationSelectDataAddress (&_PolicyDuplicationSelectData)
#else
#define _PolicyDuplicationSelectDataAddress 0
#endif // CC_PolicyDuplicationSelect

#if       CC_PolicyAuthorize
#include    "PolicyAuthorize_fp.h"

typedef TPM_RC (PolicyAuthorize_Entry)(
    PolicyAuthorize_In*         in
);


typedef const struct
{
    PolicyAuthorize_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[7];
} PolicyAuthorize_COMMAND_DESCRIPTOR_t;

PolicyAuthorize_COMMAND_DESCRIPTOR_t _PolicyAuthorizeData = {
    /* entry         */         &TPM2_PolicyAuthorize,
    /* inSize        */         (UINT16)(sizeof(PolicyAuthorize_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyAuthorize_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyAuthorize_In,
approvedPolicy)),
```

```
                                    (UINT16)(offsetof(PolicyAuthorize_In, policyRef)),
                                    (UINT16)(offsetof(PolicyAuthorize_In, keySign)),
                                    (UINT16)(offsetof(PolicyAuthorize_In, checkTicket))},
    /* types         */             {TPMI_SH_POLICY_H_UNMARSHAL,
                                    TPM2B_DIGEST_P_UNMARSHAL,
                                    TPM2B_NONCE_P_UNMARSHAL,
                                    TPM2B_NAME_P_UNMARSHAL,
                                    TPMT_TK_VERIFIED_P_UNMARSHAL,
                                    END_OF_LIST,
                                    END_OF_LIST}
};

#define _PolicyAuthorizeDataAddress (&_PolicyAuthorizeData)
#else
#define _PolicyAuthorizeDataAddress 0
#endif // CC_PolicyAuthorize

#if       CC_PolicyAuthValue
#include    "PolicyAuthValue_fp.h"

typedef TPM_RC (PolicyAuthValue_Entry)(
    PolicyAuthValue_In*         in
);


typedef const struct
{
    PolicyAuthValue_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} PolicyAuthValue_COMMAND_DESCRIPTOR_t;

PolicyAuthValue_COMMAND_DESCRIPTOR_t _PolicyAuthValueData = {
    /* entry         */         &TPM2_PolicyAuthValue,
    /* inSize        */         (UINT16)(sizeof(PolicyAuthValue_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyAuthValue_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyAuthValueDataAddress (&_PolicyAuthValueData)
#else
#define _PolicyAuthValueDataAddress 0
#endif // CC_PolicyAuthValue

#if       CC_PolicyPassword
#include    "PolicyPassword_fp.h"

typedef TPM_RC (PolicyPassword_Entry)(
    PolicyPassword_In*          in
);


typedef const struct
{
    PolicyPassword_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} PolicyPassword_COMMAND_DESCRIPTOR_t;
```

```
PolicyPassword_COMMAND_DESCRIPTOR_t _PolicyPasswordData = {
    /* entry          */         &TPM2_PolicyPassword,
    /* inSize         */         (UINT16)(sizeof(PolicyPassword_In)),
    /* outSize        */         0,
    /* offsetOfTypes  */         offsetof(PolicyPassword_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         // No parameter offsets
    /* types          */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                  END_OF_LIST,
                                  END_OF_LIST}
};

#define _PolicyPasswordDataAddress (&_PolicyPasswordData)
#else
#define _PolicyPasswordDataAddress 0
#endif // CC_PolicyPassword

#if       CC_PolicyGetDigest
#include    "PolicyGetDigest_fp.h"

typedef TPM_RC (PolicyGetDigest_Entry)(
    PolicyGetDigest_In*         in,
    PolicyGetDigest_Out*        out
);


typedef const struct
{
    PolicyGetDigest_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} PolicyGetDigest_COMMAND_DESCRIPTOR_t;

PolicyGetDigest_COMMAND_DESCRIPTOR_t _PolicyGetDigestData = {
    /* entry          */         &TPM2_PolicyGetDigest,
    /* inSize         */         (UINT16)(sizeof(PolicyGetDigest_In)),
    /* outSize        */         (UINT16)(sizeof(PolicyGetDigest_Out)),
    /* offsetOfTypes  */         offsetof(PolicyGetDigest_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         // No parameter offsets
    /* types          */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                  END_OF_LIST,
                                  TPM2B_DIGEST_P_MARSHAL,
                                  END_OF_LIST}
};

#define _PolicyGetDigestDataAddress (&_PolicyGetDigestData)
#else
#define _PolicyGetDigestDataAddress 0
#endif // CC_PolicyGetDigest

#if       CC_PolicyNvWritten
#include    "PolicyNvWritten_fp.h"

typedef TPM_RC (PolicyNvWritten_Entry)(
    PolicyNvWritten_In*         in
);


typedef const struct
{
    PolicyNvWritten_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
```

```
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyNvWritten_COMMAND_DESCRIPTOR_t;

PolicyNvWritten_COMMAND_DESCRIPTOR_t _PolicyNvWrittenData = {
    /* entry         */         &TPM2_PolicyNvWritten,
    /* inSize        */         (UINT16)(sizeof(PolicyNvWritten_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyNvWritten_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyNvWritten_In, writtenSet))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPMI_YES_NO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyNvWrittenDataAddress (&_PolicyNvWrittenData)
#else
#define _PolicyNvWrittenDataAddress 0
#endif // CC_PolicyNvWritten

#if       CC_PolicyTemplate
#include    "PolicyTemplate_fp.h"

typedef TPM_RC (PolicyTemplate_Entry)(
    PolicyTemplate_In*          in
);


typedef const struct
{
    PolicyTemplate_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyTemplate_COMMAND_DESCRIPTOR_t;

PolicyTemplate_COMMAND_DESCRIPTOR_t _PolicyTemplateData = {
    /* entry         */         &TPM2_PolicyTemplate,
    /* inSize        */         (UINT16)(sizeof(PolicyTemplate_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyTemplate_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PolicyTemplate_In, templateHash))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyTemplateDataAddress (&_PolicyTemplateData)
#else
#define _PolicyTemplateDataAddress 0
#endif // CC_PolicyTemplate

#if       CC_PolicyAuthorizeNV
#include    "PolicyAuthorizeNV_fp.h"

typedef TPM_RC (PolicyAuthorizeNV_Entry)(
    PolicyAuthorizeNV_In*       in
);


typedef const struct
{
```

```
    PolicyAuthorizeNV_Entry     *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} PolicyAuthorizeNV_COMMAND_DESCRIPTOR_t;

PolicyAuthorizeNV_COMMAND_DESCRIPTOR_t _PolicyAuthorizeNVData = {
    /* entry         */         &TPM2_PolicyAuthorizeNV,
    /* inSize        */         (UINT16)(sizeof(PolicyAuthorizeNV_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyAuthorizeNV_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PolicyAuthorizeNV_In, nvIndex)),
                                 (UINT16)(offsetof(PolicyAuthorizeNV_In,
policySession))},
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 TPMI_SH_POLICY_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyAuthorizeNVDataAddress (&_PolicyAuthorizeNVData)
#else
#define _PolicyAuthorizeNVDataAddress 0
#endif // CC_PolicyAuthorizeNV

#if       CC_PolicyCapability
#include    "PolicyCapability_fp.h"

typedef TPM_RC (PolicyCapability_Entry)(
    PolicyCapability_In*        in
);


typedef const struct
{
    PolicyCapability_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[5];
    BYTE                        types[8];
} PolicyCapability_COMMAND_DESCRIPTOR_t;

PolicyCapability_COMMAND_DESCRIPTOR_t _PolicyCapabilityData = {
    /* entry         */         &TPM2_PolicyCapability,
    /* inSize        */         (UINT16)(sizeof(PolicyCapability_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyCapability_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PolicyCapability_In, operandB)),
                                 (UINT16)(offsetof(PolicyCapability_In, offset)),
                                 (UINT16)(offsetof(PolicyCapability_In, operation)),
                                 (UINT16)(offsetof(PolicyCapability_In, capability)),
                                 (UINT16)(offsetof(PolicyCapability_In, property))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_OPERAND_P_UNMARSHAL,
                                 UINT16_P_UNMARSHAL,
                                 TPM_EO_P_UNMARSHAL,
                                 TPM_CAP_P_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
```

```
};

#define _PolicyCapabilityDataAddress (&_PolicyCapabilityData)
#else
#define _PolicyCapabilityDataAddress 0
#endif // CC_PolicyCapability

#if       CC_PolicyParameters
#include    "PolicyParameters_fp.h"

typedef TPM_RC (PolicyParameters_Entry)(
    PolicyParameters_In*        in
);


typedef const struct
{
    PolicyParameters_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} PolicyParameters_COMMAND_DESCRIPTOR_t;

PolicyParameters_COMMAND_DESCRIPTOR_t _PolicyParametersData = {
    /* entry         */         &TPM2_PolicyParameters,
    /* inSize        */         (UINT16)(sizeof(PolicyParameters_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PolicyParameters_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(PolicyParameters_In, pHash))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PolicyParametersDataAddress (&_PolicyParametersData)
#else
#define _PolicyParametersDataAddress 0
#endif // CC_PolicyParameters

#if       CC_CreatePrimary
#include    "CreatePrimary_fp.h"

typedef TPM_RC (CreatePrimary_Entry)(
    CreatePrimary_In*           in,
    CreatePrimary_Out*          out
);


typedef const struct
{
    CreatePrimary_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[9];
    BYTE                        types[13];
} CreatePrimary_COMMAND_DESCRIPTOR_t;

CreatePrimary_COMMAND_DESCRIPTOR_t _CreatePrimaryData = {
    /* entry         */         &TPM2_CreatePrimary,
    /* inSize        */         (UINT16)(sizeof(CreatePrimary_In)),
    /* outSize       */         (UINT16)(sizeof(CreatePrimary_Out)),
```

```
    /* offsetOfTypes */         offsetof(CreatePrimary_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(CreatePrimary_In, inSensitive)),
                                 (UINT16)(offsetof(CreatePrimary_In, inPublic)),
                                 (UINT16)(offsetof(CreatePrimary_In, outsideInfo)),
                                 (UINT16)(offsetof(CreatePrimary_In, creationPCR)),
                                 (UINT16)(offsetof(CreatePrimary_Out, outPublic)),
                                 (UINT16)(offsetof(CreatePrimary_Out, creationData)),
                                 (UINT16)(offsetof(CreatePrimary_Out, creationHash)),
                                 (UINT16)(offsetof(CreatePrimary_Out,
creationTicket)),
                                 (UINT16)(offsetof(CreatePrimary_Out, name))},
    /* types         */         {TPMI_RH_HIERARCHY_H_UNMARSHAL + ADD_FLAG,
                                 TPM2B_SENSITIVE_CREATE_P_UNMARSHAL,
                                 TPM2B_PUBLIC_P_UNMARSHAL,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPML_PCR_SELECTION_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM_HANDLE_H_MARSHAL,
                                 TPM2B_PUBLIC_P_MARSHAL,
                                 TPM2B_CREATION_DATA_P_MARSHAL,
                                 TPM2B_DIGEST_P_MARSHAL,
                                 TPMT_TK_CREATION_P_MARSHAL,
                                 TPM2B_NAME_P_MARSHAL,
                                 END_OF_LIST}
};

#define _CreatePrimaryDataAddress (&_CreatePrimaryData)
#else
#define _CreatePrimaryDataAddress 0
#endif // CC_CreatePrimary

#if       CC_HierarchyControl
#include    "HierarchyControl_fp.h"

typedef TPM_RC (HierarchyControl_Entry)(
    HierarchyControl_In*        in
);


typedef const struct
{
    HierarchyControl_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} HierarchyControl_COMMAND_DESCRIPTOR_t;

HierarchyControl_COMMAND_DESCRIPTOR_t _HierarchyControlData = {
    /* entry         */         &TPM2_HierarchyControl,
    /* inSize        */         (UINT16)(sizeof(HierarchyControl_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(HierarchyControl_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(HierarchyControl_In, enable)),
                                 (UINT16)(offsetof(HierarchyControl_In, state))},
    /* types         */         {TPMI_RH_HIERARCHY_H_UNMARSHAL,
                                 TPMI_RH_ENABLES_P_UNMARSHAL,
                                 TPMI_YES_NO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _HierarchyControlDataAddress (&_HierarchyControlData)
#else
```

```
#define _HierarchyControlDataAddress 0
#endif // CC_HierarchyControl

#if       CC_SetPrimaryPolicy
#include    "SetPrimaryPolicy_fp.h"

typedef TPM_RC (SetPrimaryPolicy_Entry)(
    SetPrimaryPolicy_In*        in
);


typedef const struct
{
    SetPrimaryPolicy_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} SetPrimaryPolicy_COMMAND_DESCRIPTOR_t;

SetPrimaryPolicy_COMMAND_DESCRIPTOR_t _SetPrimaryPolicyData = {
    /* entry         */         &TPM2_SetPrimaryPolicy,
    /* inSize        */         (UINT16)(sizeof(SetPrimaryPolicy_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(SetPrimaryPolicy_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(SetPrimaryPolicy_In, authPolicy)),
                                 (UINT16)(offsetof(SetPrimaryPolicy_In, hashAlg))},
    /* types         */         {TPMI_RH_HIERARCHY_POLICY_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _SetPrimaryPolicyDataAddress (&_SetPrimaryPolicyData)
#else
#define _SetPrimaryPolicyDataAddress 0
#endif // CC_SetPrimaryPolicy

#if       CC_ChangePPS
#include    "ChangePPS_fp.h"

typedef TPM_RC (ChangePPS_Entry)(
    ChangePPS_In*               in
);


typedef const struct
{
    ChangePPS_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} ChangePPS_COMMAND_DESCRIPTOR_t;

ChangePPS_COMMAND_DESCRIPTOR_t _ChangePPSData = {
    /* entry         */         &TPM2_ChangePPS,
    /* inSize        */         (UINT16)(sizeof(ChangePPS_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(ChangePPS_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 END_OF_LIST,
```

```
                                    END_OF_LIST}
};

#define _ChangePPSDataAddress (&_ChangePPSData)
#else
#define _ChangePPSDataAddress 0
#endif // CC_ChangePPS

#if       CC_ChangeEPS
#include    "ChangeEPS_fp.h"

typedef TPM_RC (ChangeEPS_Entry)(
    ChangeEPS_In*               in
);


typedef const struct
{
    ChangeEPS_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} ChangeEPS_COMMAND_DESCRIPTOR_t;

ChangeEPS_COMMAND_DESCRIPTOR_t _ChangeEPSData = {
    /* entry         */         &TPM2_ChangeEPS,
    /* inSize        */         (UINT16)(sizeof(ChangeEPS_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(ChangeEPS_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _ChangeEPSDataAddress (&_ChangeEPSData)
#else
#define _ChangeEPSDataAddress 0
#endif // CC_ChangeEPS

#if       CC_Clear
#include    "Clear_fp.h"

typedef TPM_RC (Clear_Entry)(
    Clear_In*                   in
);


typedef const struct
{
    Clear_Entry                 *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} Clear_COMMAND_DESCRIPTOR_t;

Clear_COMMAND_DESCRIPTOR_t _ClearData = {
    /* entry         */         &TPM2_Clear,
    /* inSize        */         (UINT16)(sizeof(Clear_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(Clear_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_RH_CLEAR_H_UNMARSHAL,
                                 END_OF_LIST,
```

```
                                    END_OF_LIST}
};

#define _ClearDataAddress (&_ClearData)
#else
#define _ClearDataAddress 0
#endif // CC_Clear

#if       CC_ClearControl
#include    "ClearControl_fp.h"

typedef TPM_RC (ClearControl_Entry)(
    ClearControl_In*            in
);


typedef const struct
{
    ClearControl_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} ClearControl_COMMAND_DESCRIPTOR_t;

ClearControl_COMMAND_DESCRIPTOR_t _ClearControlData = {
    /* entry         */         &TPM2_ClearControl,
    /* inSize        */         (UINT16)(sizeof(ClearControl_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(ClearControl_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ClearControl_In, disable))},
    /* types         */         {TPMI_RH_CLEAR_H_UNMARSHAL,
                                 TPMI_YES_NO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _ClearControlDataAddress (&_ClearControlData)
#else
#define _ClearControlDataAddress 0
#endif // CC_ClearControl

#if       CC_HierarchyChangeAuth
#include    "HierarchyChangeAuth_fp.h"

typedef TPM_RC (HierarchyChangeAuth_Entry)(
    HierarchyChangeAuth_In*     in
);


typedef const struct
{
    HierarchyChangeAuth_Entry   *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} HierarchyChangeAuth_COMMAND_DESCRIPTOR_t;

HierarchyChangeAuth_COMMAND_DESCRIPTOR_t _HierarchyChangeAuthData = {
    /* entry         */         &TPM2_HierarchyChangeAuth,
    /* inSize        */         (UINT16)(sizeof(HierarchyChangeAuth_In)),
    /* outSize       */         0,
```

```
    /* offsetOfTypes */         offsetof(HierarchyChangeAuth_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(HierarchyChangeAuth_In, newAuth))},
    /* types         */         {TPMI_RH_HIERARCHY_AUTH_H_UNMARSHAL,
                                 TPM2B_AUTH_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _HierarchyChangeAuthDataAddress (&_HierarchyChangeAuthData)
#else
#define _HierarchyChangeAuthDataAddress 0
#endif // CC_HierarchyChangeAuth

#if       CC_DictionaryAttackLockReset
#include    "DictionaryAttackLockReset_fp.h"

typedef TPM_RC (DictionaryAttackLockReset_Entry)(
    DictionaryAttackLockReset_In* in
);


typedef const struct
{
    DictionaryAttackLockReset_Entry *entry;
    UINT16                           inSize;
    UINT16                           outSize;
    UINT16                           offsetOfTypes;
    BYTE                             types[3];
} DictionaryAttackLockReset_COMMAND_DESCRIPTOR_t;

DictionaryAttackLockReset_COMMAND_DESCRIPTOR_t _DictionaryAttackLockResetData = {
    /* entry         */         &TPM2_DictionaryAttackLockReset,
    /* inSize        */         (UINT16)(sizeof(DictionaryAttackLockReset_In)),
    /* outSize       */         0,
    /* offsetOfTypes */
offsetof(DictionaryAttackLockReset_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_RH_LOCKOUT_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _DictionaryAttackLockResetDataAddress (&_DictionaryAttackLockResetData)
#else
#define _DictionaryAttackLockResetDataAddress 0
#endif // CC_DictionaryAttackLockReset

#if       CC_DictionaryAttackParameters
#include    "DictionaryAttackParameters_fp.h"

typedef TPM_RC (DictionaryAttackParameters_Entry)(
    DictionaryAttackParameters_In* in
);


typedef const struct
{
    DictionaryAttackParameters_Entry    *entry;
    UINT16                               inSize;
    UINT16                               outSize;
    UINT16                               offsetOfTypes;
    UINT16                               paramOffsets[3];
    BYTE                                 types[6];
} DictionaryAttackParameters_COMMAND_DESCRIPTOR_t;
```

```
DictionaryAttackParameters_COMMAND_DESCRIPTOR_t _DictionaryAttackParametersData = {
    /* entry         */         &TPM2_DictionaryAttackParameters,
    /* inSize        */         (UINT16)(sizeof(DictionaryAttackParameters_In)),
    /* outSize       */         0,
    /* offsetOfTypes */
offsetof(DictionaryAttackParameters_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(DictionaryAttackParameters_In,
newMaxTries)),
                                 (UINT16)(offsetof(DictionaryAttackParameters_In,
newRecoveryTime)),
                                 (UINT16)(offsetof(DictionaryAttackParameters_In,
lockoutRecovery))},
    /* types         */         {TPMI_RH_LOCKOUT_H_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _DictionaryAttackParametersDataAddress (&_DictionaryAttackParametersData)
#else
#define _DictionaryAttackParametersDataAddress 0
#endif // CC_DictionaryAttackParameters

#if       CC_PP_Commands
#include    "PP_Commands_fp.h"

typedef TPM_RC (PP_Commands_Entry)(
    PP_Commands_In*             in
);


typedef const struct
{
    PP_Commands_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} PP_Commands_COMMAND_DESCRIPTOR_t;

PP_Commands_COMMAND_DESCRIPTOR_t _PP_CommandsData = {
    /* entry         */         &TPM2_PP_Commands,
    /* inSize        */         (UINT16)(sizeof(PP_Commands_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(PP_Commands_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(PP_Commands_In, setList)),
                                 (UINT16)(offsetof(PP_Commands_In, clearList))},
    /* types         */         {TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 TPML_CC_P_UNMARSHAL,
                                 TPML_CC_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _PP_CommandsDataAddress (&_PP_CommandsData)
#else
#define _PP_CommandsDataAddress 0
#endif // CC_PP_Commands

#if       CC_SetAlgorithmSet
#include    "SetAlgorithmSet_fp.h"

typedef TPM_RC (SetAlgorithmSet_Entry)(
```

```
    SetAlgorithmSet_In*         in
);


typedef const struct
{
    SetAlgorithmSet_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} SetAlgorithmSet_COMMAND_DESCRIPTOR_t;

SetAlgorithmSet_COMMAND_DESCRIPTOR_t _SetAlgorithmSetData = {
    /* entry         */         &TPM2_SetAlgorithmSet,
    /* inSize        */         (UINT16)(sizeof(SetAlgorithmSet_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(SetAlgorithmSet_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(SetAlgorithmSet_In,
algorithmSet))},
    /* types         */         {TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _SetAlgorithmSetDataAddress (&_SetAlgorithmSetData)
#else
#define _SetAlgorithmSetDataAddress 0
#endif // CC_SetAlgorithmSet

#if       CC_FieldUpgradeStart
#include    "FieldUpgradeStart_fp.h"

typedef TPM_RC (FieldUpgradeStart_Entry)(
    FieldUpgradeStart_In*       in
);


typedef const struct
{
    FieldUpgradeStart_Entry     *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[6];
} FieldUpgradeStart_COMMAND_DESCRIPTOR_t;

FieldUpgradeStart_COMMAND_DESCRIPTOR_t _FieldUpgradeStartData = {
    /* entry         */         &TPM2_FieldUpgradeStart,
    /* inSize        */         (UINT16)(sizeof(FieldUpgradeStart_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(FieldUpgradeStart_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(FieldUpgradeStart_In, keyHandle)),
                                 (UINT16)(offsetof(FieldUpgradeStart_In, fuDigest)),
                                 (UINT16)(offsetof(FieldUpgradeStart_In,
manifestSignature))},
    /* types         */         {TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPM2B_DIGEST_P_UNMARSHAL,
                                 TPMT_SIGNATURE_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
```

```
};

#define _FieldUpgradeStartDataAddress (&_FieldUpgradeStartData)
#else
#define _FieldUpgradeStartDataAddress 0
#endif // CC_FieldUpgradeStart

#if       CC_FieldUpgradeData
#include    "FieldUpgradeData_fp.h"

typedef TPM_RC (FieldUpgradeData_Entry)(
    FieldUpgradeData_In*        in,
    FieldUpgradeData_Out*       out
);


typedef const struct
{
    FieldUpgradeData_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} FieldUpgradeData_COMMAND_DESCRIPTOR_t;

FieldUpgradeData_COMMAND_DESCRIPTOR_t _FieldUpgradeDataData = {
    /* entry         */         &TPM2_FieldUpgradeData,
    /* inSize        */         (UINT16)(sizeof(FieldUpgradeData_In)),
    /* outSize       */         (UINT16)(sizeof(FieldUpgradeData_Out)),
    /* offsetOfTypes */         offsetof(FieldUpgradeData_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(FieldUpgradeData_Out,
firstDigest))},
    /* types         */         {TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMT_HA_P_MARSHAL,
                                 TPMT_HA_P_MARSHAL,
                                 END_OF_LIST}
};

#define _FieldUpgradeDataDataAddress (&_FieldUpgradeDataData)
#else
#define _FieldUpgradeDataDataAddress 0
#endif // CC_FieldUpgradeData

#if       CC_FirmwareRead
#include    "FirmwareRead_fp.h"

typedef TPM_RC (FirmwareRead_Entry)(
    FirmwareRead_In*            in,
    FirmwareRead_Out*           out
);


typedef const struct
{
    FirmwareRead_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} FirmwareRead_COMMAND_DESCRIPTOR_t;

FirmwareRead_COMMAND_DESCRIPTOR_t _FirmwareReadData = {
    /* entry         */         &TPM2_FirmwareRead,
```

```
    /* inSize         */         (UINT16)(sizeof(FirmwareRead_In)),
    /* outSize        */         (UINT16)(sizeof(FirmwareRead_Out)),
    /* offsetOfTypes  */         offsetof(FirmwareRead_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         // No parameter offsets
    /* types          */         {UINT32_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_MAX_BUFFER_P_MARSHAL,
                                 END_OF_LIST}
};

#define _FirmwareReadDataAddress (&_FirmwareReadData)
#else
#define _FirmwareReadDataAddress 0
#endif // CC_FirmwareRead

#if       CC_ContextSave
#include    "ContextSave_fp.h"

typedef TPM_RC (ContextSave_Entry)(
    ContextSave_In*             in,
    ContextSave_Out*            out
);


typedef const struct
{
    ContextSave_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} ContextSave_COMMAND_DESCRIPTOR_t;

ContextSave_COMMAND_DESCRIPTOR_t _ContextSaveData = {
    /* entry          */         &TPM2_ContextSave,
    /* inSize         */         (UINT16)(sizeof(ContextSave_In)),
    /* outSize        */         (UINT16)(sizeof(ContextSave_Out)),
    /* offsetOfTypes  */         offsetof(ContextSave_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         // No parameter offsets
    /* types          */         {TPMI_DH_CONTEXT_H_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMS_CONTEXT_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ContextSaveDataAddress (&_ContextSaveData)
#else
#define _ContextSaveDataAddress 0
#endif // CC_ContextSave

#if       CC_ContextLoad
#include    "ContextLoad_fp.h"

typedef TPM_RC (ContextLoad_Entry)(
    ContextLoad_In*             in,
    ContextLoad_Out*            out
);


typedef const struct
{
    ContextLoad_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
```

```
} ContextLoad_COMMAND_DESCRIPTOR_t;

ContextLoad_COMMAND_DESCRIPTOR_t _ContextLoadData = {
    /* entry         */         &TPM2_ContextLoad,
    /* inSize        */         (UINT16)(sizeof(ContextLoad_In)),
    /* outSize       */         (UINT16)(sizeof(ContextLoad_Out)),
    /* offsetOfTypes */         offsetof(ContextLoad_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMS_CONTEXT_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMI_DH_CONTEXT_H_MARSHAL,
                                 END_OF_LIST}
};

#define _ContextLoadDataAddress (&_ContextLoadData)
#else
#define _ContextLoadDataAddress 0
#endif // CC_ContextLoad

#if       CC_FlushContext
#include    "FlushContext_fp.h"

typedef TPM_RC (FlushContext_Entry)(
    FlushContext_In*            in
);


typedef const struct
{
    FlushContext_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} FlushContext_COMMAND_DESCRIPTOR_t;

FlushContext_COMMAND_DESCRIPTOR_t _FlushContextData = {
    /* entry         */         &TPM2_FlushContext,
    /* inSize        */         (UINT16)(sizeof(FlushContext_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(FlushContext_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_DH_CONTEXT_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _FlushContextDataAddress (&_FlushContextData)
#else
#define _FlushContextDataAddress 0
#endif // CC_FlushContext

#if       CC_EvictControl
#include    "EvictControl_fp.h"

typedef TPM_RC (EvictControl_Entry)(
    EvictControl_In*            in
);


typedef const struct
{
    EvictControl_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
```

```
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} EvictControl_COMMAND_DESCRIPTOR_t;

EvictControl_COMMAND_DESCRIPTOR_t _EvictControlData = {
    /* entry         */         &TPM2_EvictControl,
    /* inSize        */         (UINT16)(sizeof(EvictControl_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(EvictControl_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(EvictControl_In, objectHandle)),
                                 (UINT16)(offsetof(EvictControl_In,
persistentHandle))},
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_DH_PERSISTENT_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _EvictControlDataAddress (&_EvictControlData)
#else
#define _EvictControlDataAddress 0
#endif // CC_EvictControl

#if       CC_ReadClock
#include    "ReadClock_fp.h"

typedef TPM_RC (ReadClock_Entry)(
    ReadClock_Out*              out
);


typedef const struct
{
    ReadClock_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} ReadClock_COMMAND_DESCRIPTOR_t;

ReadClock_COMMAND_DESCRIPTOR_t _ReadClockData = {
    /* entry         */         &TPM2_ReadClock,
    /* inSize        */         0,
    /* outSize       */         (UINT16)(sizeof(ReadClock_Out)),
    /* offsetOfTypes */         offsetof(ReadClock_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {END_OF_LIST,
                                 TPMS_TIME_INFO_P_MARSHAL,
                                 END_OF_LIST}
};

#define _ReadClockDataAddress (&_ReadClockData)
#else
#define _ReadClockDataAddress 0
#endif // CC_ReadClock

#if       CC_ClockSet
#include    "ClockSet_fp.h"

typedef TPM_RC (ClockSet_Entry)(
    ClockSet_In*                in
);


typedef const struct
```

```
{
    ClockSet_Entry              *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} ClockSet_COMMAND_DESCRIPTOR_t;

ClockSet_COMMAND_DESCRIPTOR_t _ClockSetData = {
    /* entry         */         &TPM2_ClockSet,
    /* inSize        */         (UINT16)(sizeof(ClockSet_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(ClockSet_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ClockSet_In, newTime))},
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 UINT64_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _ClockSetDataAddress (&_ClockSetData)
#else
#define _ClockSetDataAddress 0
#endif // CC_ClockSet

#if       CC_ClockRateAdjust
#include    "ClockRateAdjust_fp.h"

typedef TPM_RC (ClockRateAdjust_Entry)(
    ClockRateAdjust_In*         in
);


typedef const struct
{
    ClockRateAdjust_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} ClockRateAdjust_COMMAND_DESCRIPTOR_t;

ClockRateAdjust_COMMAND_DESCRIPTOR_t _ClockRateAdjustData = {
    /* entry         */         &TPM2_ClockRateAdjust,
    /* inSize        */         (UINT16)(sizeof(ClockRateAdjust_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(ClockRateAdjust_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ClockRateAdjust_In, rateAdjust))},
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 TPM_CLOCK_ADJUST_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _ClockRateAdjustDataAddress (&_ClockRateAdjustData)
#else
#define _ClockRateAdjustDataAddress 0
#endif // CC_ClockRateAdjust

#if       CC_GetCapability
#include    "GetCapability_fp.h"

typedef TPM_RC (GetCapability_Entry)(
    GetCapability_In*           in,
```

```
    GetCapability_Out*          out
);


typedef const struct
{
    GetCapability_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} GetCapability_COMMAND_DESCRIPTOR_t;

GetCapability_COMMAND_DESCRIPTOR_t _GetCapabilityData = {
    /* entry         */         &TPM2_GetCapability,
    /* inSize        */         (UINT16)(sizeof(GetCapability_In)),
    /* outSize       */         (UINT16)(sizeof(GetCapability_Out)),
    /* offsetOfTypes */         offsetof(GetCapability_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(GetCapability_In, property)),
                                 (UINT16)(offsetof(GetCapability_In, propertyCount)),
                                 (UINT16)(offsetof(GetCapability_Out,
capabilityData))},
    /* types         */         {TPM_CAP_P_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMI_YES_NO_P_MARSHAL,
                                 TPMS_CAPABILITY_DATA_P_MARSHAL,
                                 END_OF_LIST}
};

#define _GetCapabilityDataAddress (&_GetCapabilityData)
#else
#define _GetCapabilityDataAddress 0
#endif // CC_GetCapability

#if       CC_TestParms
#include    "TestParms_fp.h"

typedef TPM_RC (TestParms_Entry)(
    TestParms_In*               in
);


typedef const struct
{
    TestParms_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} TestParms_COMMAND_DESCRIPTOR_t;

TestParms_COMMAND_DESCRIPTOR_t _TestParmsData = {
    /* entry         */         &TPM2_TestParms,
    /* inSize        */         (UINT16)(sizeof(TestParms_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(TestParms_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMT_PUBLIC_PARMS_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _TestParmsDataAddress (&_TestParmsData)
```

```
#else
#define _TestParmsDataAddress 0
#endif // CC_TestParms

#if       CC_NV_DefineSpace
#include    "NV_DefineSpace_fp.h"

typedef TPM_RC (NV_DefineSpace_Entry)(
    NV_DefineSpace_In*          in
);


typedef const struct
{
    NV_DefineSpace_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} NV_DefineSpace_COMMAND_DESCRIPTOR_t;

NV_DefineSpace_COMMAND_DESCRIPTOR_t _NV_DefineSpaceData = {
    /* entry         */         &TPM2_NV_DefineSpace,
    /* inSize        */         (UINT16)(sizeof(NV_DefineSpace_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_DefineSpace_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_DefineSpace_In, auth)),
                                 (UINT16)(offsetof(NV_DefineSpace_In, publicInfo))},
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 TPM2B_AUTH_P_UNMARSHAL,
                                 TPM2B_NV_PUBLIC_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_DefineSpaceDataAddress (&_NV_DefineSpaceData)
#else
#define _NV_DefineSpaceDataAddress 0
#endif // CC_NV_DefineSpace

#if       CC_NV_UndefineSpace
#include    "NV_UndefineSpace_fp.h"

typedef TPM_RC (NV_UndefineSpace_Entry)(
    NV_UndefineSpace_In*        in
);


typedef const struct
{
    NV_UndefineSpace_Entry      *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} NV_UndefineSpace_COMMAND_DESCRIPTOR_t;

NV_UndefineSpace_COMMAND_DESCRIPTOR_t _NV_UndefineSpaceData = {
    /* entry         */         &TPM2_NV_UndefineSpace,
    /* inSize        */         (UINT16)(sizeof(NV_UndefineSpace_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_UndefineSpace_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(NV_UndefineSpace_In, nvIndex))},
```

```
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 TPMI_RH_NV_DEFINED_INDEX_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_UndefineSpaceDataAddress (&_NV_UndefineSpaceData)
#else
#define _NV_UndefineSpaceDataAddress 0
#endif // CC_NV_UndefineSpace

#if       CC_NV_UndefineSpaceSpecial
#include    "NV_UndefineSpaceSpecial_fp.h"

typedef TPM_RC (NV_UndefineSpaceSpecial_Entry)(
    NV_UndefineSpaceSpecial_In* in
);


typedef const struct
{
    NV_UndefineSpaceSpecial_Entry   *entry;
    UINT16                          inSize;
    UINT16                          outSize;
    UINT16                          offsetOfTypes;
    UINT16                          paramOffsets[1];
    BYTE                            types[4];
} NV_UndefineSpaceSpecial_COMMAND_DESCRIPTOR_t;

NV_UndefineSpaceSpecial_COMMAND_DESCRIPTOR_t _NV_UndefineSpaceSpecialData = {
    /* entry         */         &TPM2_NV_UndefineSpaceSpecial,
    /* inSize        */         (UINT16)(sizeof(NV_UndefineSpaceSpecial_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_UndefineSpaceSpecial_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(NV_UndefineSpaceSpecial_In,
platform))},
    /* types         */         {TPMI_RH_NV_DEFINED_INDEX_H_UNMARSHAL,
                                 TPMI_RH_PLATFORM_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_UndefineSpaceSpecialDataAddress (&_NV_UndefineSpaceSpecialData)
#else
#define _NV_UndefineSpaceSpecialDataAddress 0
#endif // CC_NV_UndefineSpaceSpecial

#if       CC_NV_ReadPublic
#include    "NV_ReadPublic_fp.h"

typedef TPM_RC (NV_ReadPublic_Entry)(
    NV_ReadPublic_In*           in,
    NV_ReadPublic_Out*          out
);


typedef const struct
{
    NV_ReadPublic_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} NV_ReadPublic_COMMAND_DESCRIPTOR_t;
```

```
NV_ReadPublic_COMMAND_DESCRIPTOR_t _NV_ReadPublicData = {
    /* entry          */         &TPM2_NV_ReadPublic,
    /* inSize         */         (UINT16)(sizeof(NV_ReadPublic_In)),
    /* outSize        */         (UINT16)(sizeof(NV_ReadPublic_Out)),
    /* offsetOfTypes */          offsetof(NV_ReadPublic_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         {(UINT16)(offsetof(NV_ReadPublic_Out, nvName))},
    /* types          */         {TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                  END_OF_LIST,
                                  TPM2B_NV_PUBLIC_P_MARSHAL,
                                  TPM2B_NAME_P_MARSHAL,
                                  END_OF_LIST}
};

#define _NV_ReadPublicDataAddress (&_NV_ReadPublicData)
#else
#define _NV_ReadPublicDataAddress 0
#endif // CC_NV_ReadPublic

#if       CC_NV_Write
#include    "NV_Write_fp.h"

typedef TPM_RC (NV_Write_Entry)(
    NV_Write_In*                in
);


typedef const struct
{
    NV_Write_Entry              *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[6];
} NV_Write_COMMAND_DESCRIPTOR_t;

NV_Write_COMMAND_DESCRIPTOR_t _NV_WriteData = {
    /* entry          */         &TPM2_NV_Write,
    /* inSize         */         (UINT16)(sizeof(NV_Write_In)),
    /* outSize        */         0,
    /* offsetOfTypes */          offsetof(NV_Write_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         {(UINT16)(offsetof(NV_Write_In, nvIndex)),
                                  (UINT16)(offsetof(NV_Write_In, data)),
                                  (UINT16)(offsetof(NV_Write_In, offset))},
    /* types          */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                  TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                  TPM2B_MAX_NV_BUFFER_P_UNMARSHAL,
                                  UINT16_P_UNMARSHAL,
                                  END_OF_LIST,
                                  END_OF_LIST}
};

#define _NV_WriteDataAddress (&_NV_WriteData)
#else
#define _NV_WriteDataAddress 0
#endif // CC_NV_Write

#if       CC_NV_Increment
#include    "NV_Increment_fp.h"

typedef TPM_RC (NV_Increment_Entry)(
    NV_Increment_In*            in
);
```

```
typedef const struct
{
    NV_Increment_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} NV_Increment_COMMAND_DESCRIPTOR_t;

NV_Increment_COMMAND_DESCRIPTOR_t _NV_IncrementData = {
    /* entry         */         &TPM2_NV_Increment,
    /* inSize        */         (UINT16)(sizeof(NV_Increment_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_Increment_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_Increment_In, nvIndex))},
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_IncrementDataAddress (&_NV_IncrementData)
#else
#define _NV_IncrementDataAddress 0
#endif // CC_NV_Increment

#if       CC_NV_Extend
#include    "NV_Extend_fp.h"

typedef TPM_RC (NV_Extend_Entry)(
    NV_Extend_In*               in
);


typedef const struct
{
    NV_Extend_Entry             *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} NV_Extend_COMMAND_DESCRIPTOR_t;

NV_Extend_COMMAND_DESCRIPTOR_t _NV_ExtendData = {
    /* entry         */         &TPM2_NV_Extend,
    /* inSize        */         (UINT16)(sizeof(NV_Extend_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_Extend_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_Extend_In, nvIndex)),
                                 (UINT16)(offsetof(NV_Extend_In, data))},
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 TPM2B_MAX_NV_BUFFER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_ExtendDataAddress (&_NV_ExtendData)
#else
#define _NV_ExtendDataAddress 0
#endif // CC_NV_Extend

#if       CC_NV_SetBits
#include    "NV_SetBits_fp.h"
```

```
typedef TPM_RC (NV_SetBits_Entry)(
    NV_SetBits_In*              in
);


typedef const struct
{
    NV_SetBits_Entry            *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} NV_SetBits_COMMAND_DESCRIPTOR_t;

NV_SetBits_COMMAND_DESCRIPTOR_t _NV_SetBitsData = {
    /* entry         */         &TPM2_NV_SetBits,
    /* inSize        */         (UINT16)(sizeof(NV_SetBits_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_SetBits_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_SetBits_In, nvIndex)),
                                 (UINT16)(offsetof(NV_SetBits_In, bits))},
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 UINT64_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_SetBitsDataAddress (&_NV_SetBitsData)
#else
#define _NV_SetBitsDataAddress 0
#endif // CC_NV_SetBits

#if       CC_NV_WriteLock
#include    "NV_WriteLock_fp.h"

typedef TPM_RC (NV_WriteLock_Entry)(
    NV_WriteLock_In*            in
);


typedef const struct
{
    NV_WriteLock_Entry          *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} NV_WriteLock_COMMAND_DESCRIPTOR_t;

NV_WriteLock_COMMAND_DESCRIPTOR_t _NV_WriteLockData = {
    /* entry         */         &TPM2_NV_WriteLock,
    /* inSize        */         (UINT16)(sizeof(NV_WriteLock_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_WriteLock_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_WriteLock_In, nvIndex))},
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_WriteLockDataAddress (&_NV_WriteLockData)
```

```
#else
#define _NV_WriteLockDataAddress 0
#endif // CC_NV_WriteLock

#if       CC_NV_GlobalWriteLock
#include    "NV_GlobalWriteLock_fp.h"

typedef TPM_RC (NV_GlobalWriteLock_Entry)(
    NV_GlobalWriteLock_In*      in
);


typedef const struct
{
    NV_GlobalWriteLock_Entry    *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[3];
} NV_GlobalWriteLock_COMMAND_DESCRIPTOR_t;

NV_GlobalWriteLock_COMMAND_DESCRIPTOR_t _NV_GlobalWriteLockData = {
    /* entry         */         &TPM2_NV_GlobalWriteLock,
    /* inSize        */         (UINT16)(sizeof(NV_GlobalWriteLock_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_GlobalWriteLock_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_GlobalWriteLockDataAddress (&_NV_GlobalWriteLockData)
#else
#define _NV_GlobalWriteLockDataAddress 0
#endif // CC_NV_GlobalWriteLock

#if       CC_NV_Read
#include    "NV_Read_fp.h"

typedef TPM_RC (NV_Read_Entry)(
    NV_Read_In*                 in,
    NV_Read_Out*                out
);


typedef const struct
{
    NV_Read_Entry               *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} NV_Read_COMMAND_DESCRIPTOR_t;

NV_Read_COMMAND_DESCRIPTOR_t _NV_ReadData = {
    /* entry         */         &TPM2_NV_Read,
    /* inSize        */         (UINT16)(sizeof(NV_Read_In)),
    /* outSize       */         (UINT16)(sizeof(NV_Read_Out)),
    /* offsetOfTypes */         offsetof(NV_Read_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_Read_In, nvIndex)),
                                 (UINT16)(offsetof(NV_Read_In, size)),
                                 (UINT16)(offsetof(NV_Read_In, offset))},
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
```

```
                                TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                UINT16_P_UNMARSHAL,
                                UINT16_P_UNMARSHAL,
                                END_OF_LIST,
                                TPM2B_MAX_NV_BUFFER_P_MARSHAL,
                                END_OF_LIST}
};

#define _NV_ReadDataAddress (&_NV_ReadData)
#else
#define _NV_ReadDataAddress 0
#endif // CC_NV_Read

#if       CC_NV_ReadLock
#include    "NV_ReadLock_fp.h"

typedef TPM_RC (NV_ReadLock_Entry)(
    NV_ReadLock_In*             in
);


typedef const struct
{
    NV_ReadLock_Entry           *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} NV_ReadLock_COMMAND_DESCRIPTOR_t;

NV_ReadLock_COMMAND_DESCRIPTOR_t _NV_ReadLockData = {
    /* entry         */         &TPM2_NV_ReadLock,
    /* inSize        */         (UINT16)(sizeof(NV_ReadLock_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_ReadLock_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_ReadLock_In, nvIndex))},
    /* types         */         {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_ReadLockDataAddress (&_NV_ReadLockData)
#else
#define _NV_ReadLockDataAddress 0
#endif // CC_NV_ReadLock

#if       CC_NV_ChangeAuth
#include    "NV_ChangeAuth_fp.h"

typedef TPM_RC (NV_ChangeAuth_Entry)(
    NV_ChangeAuth_In*           in
);


typedef const struct
{
    NV_ChangeAuth_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} NV_ChangeAuth_COMMAND_DESCRIPTOR_t;
```

```
NV_ChangeAuth_COMMAND_DESCRIPTOR_t _NV_ChangeAuthData = {
    /* entry          */         &TPM2_NV_ChangeAuth,
    /* inSize         */         (UINT16)(sizeof(NV_ChangeAuth_In)),
    /* outSize        */         0,
    /* offsetOfTypes */          offsetof(NV_ChangeAuth_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         {(UINT16)(offsetof(NV_ChangeAuth_In, newAuth))},
    /* types          */         {TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 TPM2B_AUTH_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_ChangeAuthDataAddress (&_NV_ChangeAuthData)
#else
#define _NV_ChangeAuthDataAddress 0
#endif // CC_NV_ChangeAuth

#if       CC_NV_Certify
#include    "NV_Certify_fp.h"

typedef TPM_RC (NV_Certify_Entry)(
    NV_Certify_In*              in,
    NV_Certify_Out*             out
);


typedef const struct
{
    NV_Certify_Entry            *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[7];
    BYTE                        types[11];
} NV_Certify_COMMAND_DESCRIPTOR_t;

NV_Certify_COMMAND_DESCRIPTOR_t _NV_CertifyData = {
    /* entry          */         &TPM2_NV_Certify,
    /* inSize         */         (UINT16)(sizeof(NV_Certify_In)),
    /* outSize        */         (UINT16)(sizeof(NV_Certify_Out)),
    /* offsetOfTypes */          offsetof(NV_Certify_COMMAND_DESCRIPTOR_t, types),
    /* offsets        */         {(UINT16)(offsetof(NV_Certify_In, authHandle)),
                                 (UINT16)(offsetof(NV_Certify_In, nvIndex)),
                                 (UINT16)(offsetof(NV_Certify_In, qualifyingData)),
                                 (UINT16)(offsetof(NV_Certify_In, inScheme)),
                                 (UINT16)(offsetof(NV_Certify_In, size)),
                                 (UINT16)(offsetof(NV_Certify_In, offset)),
                                 (UINT16)(offsetof(NV_Certify_Out, signature))},
    /* types          */         {TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
                                 TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 TPM2B_DATA_P_UNMARSHAL,
                                 TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
                                 UINT16_P_UNMARSHAL,
                                 UINT16_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_ATTEST_P_MARSHAL,
                                 TPMT_SIGNATURE_P_MARSHAL,
                                 END_OF_LIST}
};

#define _NV_CertifyDataAddress (&_NV_CertifyData)
#else
#define _NV_CertifyDataAddress 0
#endif // CC_NV_Certify
```

```
#if       CC_NV_DefineSpace2
#include    "NV_DefineSpace2_fp.h"

typedef TPM_RC (NV_DefineSpace2_Entry)(
    NV_DefineSpace2_In*         in
);


typedef const struct
{
    NV_DefineSpace2_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} NV_DefineSpace2_COMMAND_DESCRIPTOR_t;

NV_DefineSpace2_COMMAND_DESCRIPTOR_t _NV_DefineSpace2Data = {
    /* entry         */         &TPM2_NV_DefineSpace2,
    /* inSize        */         (UINT16)(sizeof(NV_DefineSpace2_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(NV_DefineSpace2_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_DefineSpace2_In, auth)),
                                 (UINT16)(offsetof(NV_DefineSpace2_In, publicInfo))},
    /* types         */         {TPMI_RH_PROVISION_H_UNMARSHAL,
                                 TPM2B_AUTH_P_UNMARSHAL,
                                 TPM2B_NV_PUBLIC_2_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _NV_DefineSpace2DataAddress (&_NV_DefineSpace2Data)
#else
#define _NV_DefineSpace2DataAddress 0
#endif // CC_NV_DefineSpace2

#if       CC_NV_ReadPublic2
#include    "NV_ReadPublic2_fp.h"

typedef TPM_RC (NV_ReadPublic2_Entry)(
    NV_ReadPublic2_In*          in,
    NV_ReadPublic2_Out*         out
);


typedef const struct
{
    NV_ReadPublic2_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[5];
} NV_ReadPublic2_COMMAND_DESCRIPTOR_t;

NV_ReadPublic2_COMMAND_DESCRIPTOR_t _NV_ReadPublic2Data = {
    /* entry         */         &TPM2_NV_ReadPublic2,
    /* inSize        */         (UINT16)(sizeof(NV_ReadPublic2_In)),
    /* outSize       */         (UINT16)(sizeof(NV_ReadPublic2_Out)),
    /* offsetOfTypes */         offsetof(NV_ReadPublic2_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(NV_ReadPublic2_Out, nvName))},
    /* types         */         {TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_NV_PUBLIC_2_P_MARSHAL,
                                 TPM2B_NAME_P_MARSHAL,
```

```
                                    END_OF_LIST}
};

#define _NV_ReadPublic2DataAddress (&_NV_ReadPublic2Data)
#else
#define _NV_ReadPublic2DataAddress 0
#endif // CC_NV_ReadPublic2

#if       CC_SetCapability
#include    "SetCapability_fp.h"

typedef TPM_RC (SetCapability_Entry)(
    SetCapability_In*        in
);

typedef const struct
{
    SetCapability_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} SetCapability_COMMAND_DESCRIPTOR_t;

SetCapability_COMMAND_DESCRIPTOR_t _SetCapabilityData = {
    /* entry         */         &TPM2_SetCapability,
    /* inSize        */         (UINT16)(sizeof(SetCapability_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(SetCapability_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(SetCapability_In,
setCapabilityData))},
    /* types         */         {TPMI_RH_HIERARCHY_H_UNMARSHAL,
                                 TPM2B_SET_CAPABILITY_DATA_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _SetCapabilityDataAddress (&_SetCapabilityData)
#else
#define _SetCapabilityDataAddress 0
#endif // CC_SetCapability

#if       CC_AC_Send
#include    "AC_Send_fp.h"

typedef TPM_RC (AC_Send_Entry)(
    AC_Send_In*                 in,
    AC_Send_Out*                out
);


typedef const struct
{
    AC_Send_Entry               *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[3];
    BYTE                        types[7];
} AC_Send_COMMAND_DESCRIPTOR_t;

AC_Send_COMMAND_DESCRIPTOR_t _AC_SendData = {
    /* entry         */         &TPM2_AC_Send,
    /* inSize        */         (UINT16)(sizeof(AC_Send_In)),
    /* outSize       */         (UINT16)(sizeof(AC_Send_Out)),
```

```
    /* offsetOfTypes */         offsetof(AC_Send_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(AC_Send_In, authHandle)),
                                 (UINT16)(offsetof(AC_Send_In, ac)),
                                 (UINT16)(offsetof(AC_Send_In, acDataIn))},
    /* types         */         {TPMI_DH_OBJECT_H_UNMARSHAL,
                                 TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                 TPMI_RH_AC_H_UNMARSHAL,
                                 TPM2B_MAX_BUFFER_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPMS_AC_OUTPUT_P_MARSHAL,
                                 END_OF_LIST}
};

#define _AC_SendDataAddress (&_AC_SendData)
#else
#define _AC_SendDataAddress 0
#endif // CC_AC_Send

#if       CC_Policy_AC_SendSelect
#include    "Policy_AC_SendSelect_fp.h"

typedef TPM_RC (Policy_AC_SendSelect_Entry)(
    Policy_AC_SendSelect_In*    in
);


typedef const struct
{
    Policy_AC_SendSelect_Entry  *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[4];
    BYTE                        types[7];
} Policy_AC_SendSelect_COMMAND_DESCRIPTOR_t;

Policy_AC_SendSelect_COMMAND_DESCRIPTOR_t _Policy_AC_SendSelectData = {
    /* entry         */         &TPM2_Policy_AC_SendSelect,
    /* inSize        */         (UINT16)(sizeof(Policy_AC_SendSelect_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(Policy_AC_SendSelect_COMMAND_DESCRIPTOR_t,
types),
    /* offsets       */         {(UINT16)(offsetof(Policy_AC_SendSelect_In,
objectName)),
                                 (UINT16)(offsetof(Policy_AC_SendSelect_In,
authHandleName)),
                                 (UINT16)(offsetof(Policy_AC_SendSelect_In, acName)),
                                 (UINT16)(offsetof(Policy_AC_SendSelect_In,
includeObject))},
    /* types         */         {TPMI_SH_POLICY_H_UNMARSHAL,
                                 TPM2B_NAME_P_UNMARSHAL,
                                 TPM2B_NAME_P_UNMARSHAL,
                                 TPM2B_NAME_P_UNMARSHAL,
                                 TPMI_YES_NO_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _Policy_AC_SendSelectDataAddress (&_Policy_AC_SendSelectData)
#else
#define _Policy_AC_SendSelectDataAddress 0
#endif // CC_Policy_AC_SendSelect

#if       CC_ACT_SetTimeout
#include    "ACT_SetTimeout_fp.h"
```

```
typedef TPM_RC (ACT_SetTimeout_Entry)(
    ACT_SetTimeout_In*          in
);


typedef const struct
{
    ACT_SetTimeout_Entry        *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} ACT_SetTimeout_COMMAND_DESCRIPTOR_t;

ACT_SetTimeout_COMMAND_DESCRIPTOR_t _ACT_SetTimeoutData = {
    /* entry         */         &TPM2_ACT_SetTimeout,
    /* inSize        */         (UINT16)(sizeof(ACT_SetTimeout_In)),
    /* outSize       */         0,
    /* offsetOfTypes */         offsetof(ACT_SetTimeout_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         {(UINT16)(offsetof(ACT_SetTimeout_In, startTimeout))},
    /* types         */         {TPMI_RH_ACT_H_UNMARSHAL,
                                 UINT32_P_UNMARSHAL,
                                 END_OF_LIST,
                                 END_OF_LIST}
};

#define _ACT_SetTimeoutDataAddress (&_ACT_SetTimeoutData)
#else
#define _ACT_SetTimeoutDataAddress 0
#endif // CC_ACT_SetTimeout

#if       CC_Vendor_TCG_Test
#include    "Vendor_TCG_Test_fp.h"

typedef TPM_RC (Vendor_TCG_Test_Entry)(
    Vendor_TCG_Test_In*         in,
    Vendor_TCG_Test_Out*        out
);


typedef const struct
{
    Vendor_TCG_Test_Entry       *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    BYTE                        types[4];
} Vendor_TCG_Test_COMMAND_DESCRIPTOR_t;

Vendor_TCG_Test_COMMAND_DESCRIPTOR_t _Vendor_TCG_TestData = {
    /* entry         */         &TPM2_Vendor_TCG_Test,
    /* inSize        */         (UINT16)(sizeof(Vendor_TCG_Test_In)),
    /* outSize       */         (UINT16)(sizeof(Vendor_TCG_Test_Out)),
    /* offsetOfTypes */         offsetof(Vendor_TCG_Test_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */         // No parameter offsets
    /* types         */         {TPM2B_DATA_P_UNMARSHAL,
                                 END_OF_LIST,
                                 TPM2B_DATA_P_MARSHAL,
                                 END_OF_LIST}
};

#define _Vendor_TCG_TestDataAddress (&_Vendor_TCG_TestData)
#else
#define _Vendor_TCG_TestDataAddress 0
#endif // CC_Vendor_TCG_Test
```

```
// Lookup table to access the per-command tables above

COMMAND_DESCRIPTOR_t* s_CommandDataArray[] = {
#if (PAD_LIST || CC_NV_UndefineSpaceSpecial)
        (COMMAND_DESCRIPTOR_t*)_NV_UndefineSpaceSpecialDataAddress,
#endif // CC_NV_UndefineSpaceSpecial
#if (PAD_LIST || CC_EvictControl)
        (COMMAND_DESCRIPTOR_t*)_EvictControlDataAddress,
#endif // CC_EvictControl
#if (PAD_LIST || CC_HierarchyControl)
        (COMMAND_DESCRIPTOR_t*)_HierarchyControlDataAddress,
#endif // CC_HierarchyControl
#if (PAD_LIST || CC_NV_UndefineSpace)
        (COMMAND_DESCRIPTOR_t*)_NV_UndefineSpaceDataAddress,
#endif // CC_NV_UndefineSpace
#if (PAD_LIST)
       (COMMAND_DESCRIPTOR_t*)0,
#endif //
#if (PAD_LIST || CC_ChangeEPS)
        (COMMAND_DESCRIPTOR_t*)_ChangeEPSDataAddress,
#endif // CC_ChangeEPS
#if (PAD_LIST || CC_ChangePPS)
        (COMMAND_DESCRIPTOR_t*)_ChangePPSDataAddress,
#endif // CC_ChangePPS
#if (PAD_LIST || CC_Clear)
        (COMMAND_DESCRIPTOR_t*)_ClearDataAddress,
#endif // CC_Clear
#if (PAD_LIST || CC_ClearControl)
        (COMMAND_DESCRIPTOR_t*)_ClearControlDataAddress,
#endif // CC_ClearControl
#if (PAD_LIST || CC_ClockSet)
        (COMMAND_DESCRIPTOR_t*)_ClockSetDataAddress,
#endif // CC_ClockSet
#if (PAD_LIST || CC_HierarchyChangeAuth)
        (COMMAND_DESCRIPTOR_t*)_HierarchyChangeAuthDataAddress,
#endif // CC_HierarchyChangeAuth
#if (PAD_LIST || CC_NV_DefineSpace)
        (COMMAND_DESCRIPTOR_t*)_NV_DefineSpaceDataAddress,
#endif // CC_NV_DefineSpace
#if (PAD_LIST || CC_PCR_Allocate)
        (COMMAND_DESCRIPTOR_t*)_PCR_AllocateDataAddress,
#endif // CC_PCR_Allocate
#if (PAD_LIST || CC_PCR_SetAuthPolicy)
        (COMMAND_DESCRIPTOR_t*)_PCR_SetAuthPolicyDataAddress,
#endif // CC_PCR_SetAuthPolicy
#if (PAD_LIST || CC_PP_Commands)
        (COMMAND_DESCRIPTOR_t*)_PP_CommandsDataAddress,
#endif // CC_PP_Commands
#if (PAD_LIST || CC_SetPrimaryPolicy)
        (COMMAND_DESCRIPTOR_t*)_SetPrimaryPolicyDataAddress,
#endif // CC_SetPrimaryPolicy
#if (PAD_LIST || CC_FieldUpgradeStart)
        (COMMAND_DESCRIPTOR_t*)_FieldUpgradeStartDataAddress,
#endif // CC_FieldUpgradeStart
#if (PAD_LIST || CC_ClockRateAdjust)
        (COMMAND_DESCRIPTOR_t*)_ClockRateAdjustDataAddress,
#endif // CC_ClockRateAdjust
#if (PAD_LIST || CC_CreatePrimary)
        (COMMAND_DESCRIPTOR_t*)_CreatePrimaryDataAddress,
#endif // CC_CreatePrimary
#if (PAD_LIST || CC_NV_GlobalWriteLock)
        (COMMAND_DESCRIPTOR_t*)_NV_GlobalWriteLockDataAddress,
#endif // CC_NV_GlobalWriteLock
#if (PAD_LIST || CC_GetCommandAuditDigest)
```

```
        (COMMAND_DESCRIPTOR_t*)_GetCommandAuditDigestDataAddress,
#endif // CC_GetCommandAuditDigest
#if (PAD_LIST || CC_NV_Increment)
        (COMMAND_DESCRIPTOR_t*)_NV_IncrementDataAddress,
#endif // CC_NV_Increment
#if (PAD_LIST || CC_NV_SetBits)
        (COMMAND_DESCRIPTOR_t*)_NV_SetBitsDataAddress,
#endif // CC_NV_SetBits
#if (PAD_LIST || CC_NV_Extend)
        (COMMAND_DESCRIPTOR_t*)_NV_ExtendDataAddress,
#endif // CC_NV_Extend
#if (PAD_LIST || CC_NV_Write)
        (COMMAND_DESCRIPTOR_t*)_NV_WriteDataAddress,
#endif // CC_NV_Write
#if (PAD_LIST || CC_NV_WriteLock)
        (COMMAND_DESCRIPTOR_t*)_NV_WriteLockDataAddress,
#endif // CC_NV_WriteLock
#if (PAD_LIST || CC_DictionaryAttackLockReset)
        (COMMAND_DESCRIPTOR_t*)_DictionaryAttackLockResetDataAddress,
#endif // CC_DictionaryAttackLockReset
#if (PAD_LIST || CC_DictionaryAttackParameters)
        (COMMAND_DESCRIPTOR_t*)_DictionaryAttackParametersDataAddress,
#endif // CC_DictionaryAttackParameters
#if (PAD_LIST || CC_NV_ChangeAuth)
        (COMMAND_DESCRIPTOR_t*)_NV_ChangeAuthDataAddress,
#endif // CC_NV_ChangeAuth
#if (PAD_LIST || CC_PCR_Event)
        (COMMAND_DESCRIPTOR_t*)_PCR_EventDataAddress,
#endif // CC_PCR_Event
#if (PAD_LIST || CC_PCR_Reset)
        (COMMAND_DESCRIPTOR_t*)_PCR_ResetDataAddress,
#endif // CC_PCR_Reset
#if (PAD_LIST || CC_SequenceComplete)
        (COMMAND_DESCRIPTOR_t*)_SequenceCompleteDataAddress,
#endif // CC_SequenceComplete
#if (PAD_LIST || CC_SetAlgorithmSet)
        (COMMAND_DESCRIPTOR_t*)_SetAlgorithmSetDataAddress,
#endif // CC_SetAlgorithmSet
#if (PAD_LIST || CC_SetCommandCodeAuditStatus)
        (COMMAND_DESCRIPTOR_t*)_SetCommandCodeAuditStatusDataAddress,
#endif // CC_SetCommandCodeAuditStatus
#if (PAD_LIST || CC_FieldUpgradeData)
        (COMMAND_DESCRIPTOR_t*)_FieldUpgradeDataDataAddress,
#endif // CC_FieldUpgradeData
#if (PAD_LIST || CC_IncrementalSelfTest)
        (COMMAND_DESCRIPTOR_t*)_IncrementalSelfTestDataAddress,
#endif // CC_IncrementalSelfTest
#if (PAD_LIST || CC_SelfTest)
        (COMMAND_DESCRIPTOR_t*)_SelfTestDataAddress,
#endif // CC_SelfTest
#if (PAD_LIST || CC_Startup)
        (COMMAND_DESCRIPTOR_t*)_StartupDataAddress,
#endif // CC_Startup
#if (PAD_LIST || CC_Shutdown)
        (COMMAND_DESCRIPTOR_t*)_ShutdownDataAddress,
#endif // CC_Shutdown
#if (PAD_LIST || CC_StirRandom)
        (COMMAND_DESCRIPTOR_t*)_StirRandomDataAddress,
#endif // CC_StirRandom
#if (PAD_LIST || CC_ActivateCredential)
        (COMMAND_DESCRIPTOR_t*)_ActivateCredentialDataAddress,
#endif // CC_ActivateCredential
#if (PAD_LIST || CC_Certify)
        (COMMAND_DESCRIPTOR_t*)_CertifyDataAddress,
#endif // CC_Certify
#if (PAD_LIST || CC_PolicyNV)
```

```
        (COMMAND_DESCRIPTOR_t*)_PolicyNVDataAddress,
#endif // CC_PolicyNV
#if (PAD_LIST || CC_CertifyCreation)
        (COMMAND_DESCRIPTOR_t*)_CertifyCreationDataAddress,
#endif // CC_CertifyCreation
#if (PAD_LIST || CC_Duplicate)
        (COMMAND_DESCRIPTOR_t*)_DuplicateDataAddress,
#endif // CC_Duplicate
#if (PAD_LIST || CC_GetTime)
        (COMMAND_DESCRIPTOR_t*)_GetTimeDataAddress,
#endif // CC_GetTime
#if (PAD_LIST || CC_GetSessionAuditDigest)
        (COMMAND_DESCRIPTOR_t*)_GetSessionAuditDigestDataAddress,
#endif // CC_GetSessionAuditDigest
#if (PAD_LIST || CC_NV_Read)
        (COMMAND_DESCRIPTOR_t*)_NV_ReadDataAddress,
#endif // CC_NV_Read
#if (PAD_LIST || CC_NV_ReadLock)
        (COMMAND_DESCRIPTOR_t*)_NV_ReadLockDataAddress,
#endif // CC_NV_ReadLock
#if (PAD_LIST || CC_ObjectChangeAuth)
        (COMMAND_DESCRIPTOR_t*)_ObjectChangeAuthDataAddress,
#endif // CC_ObjectChangeAuth
#if (PAD_LIST || CC_PolicySecret)
        (COMMAND_DESCRIPTOR_t*)_PolicySecretDataAddress,
#endif // CC_PolicySecret
#if (PAD_LIST || CC_Rewrap)
        (COMMAND_DESCRIPTOR_t*)_RewrapDataAddress,
#endif // CC_Rewrap
#if (PAD_LIST || CC_Create)
        (COMMAND_DESCRIPTOR_t*)_CreateDataAddress,
#endif // CC_Create
#if (PAD_LIST || CC_ECDH_ZGen)
        (COMMAND_DESCRIPTOR_t*)_ECDH_ZGenDataAddress,
#endif // CC_ECDH_ZGen
#if (PAD_LIST || (CC_HMAC || CC_MAC))
#   if CC_HMAC
        (COMMAND_DESCRIPTOR_t*)_HMACDataAddress,
#   endif
#   if CC_MAC
        (COMMAND_DESCRIPTOR_t*)_MACDataAddress,
#   endif
#endif // (CC_HMAC || CC_MAC)
#if (PAD_LIST || CC_Import)
        (COMMAND_DESCRIPTOR_t*)_ImportDataAddress,
#endif // CC_Import
#if (PAD_LIST || CC_Load)
        (COMMAND_DESCRIPTOR_t*)_LoadDataAddress,
#endif // CC_Load
#if (PAD_LIST || CC_Quote)
        (COMMAND_DESCRIPTOR_t*)_QuoteDataAddress,
#endif // CC_Quote
#if (PAD_LIST || CC_RSA_Decrypt)
        (COMMAND_DESCRIPTOR_t*)_RSA_DecryptDataAddress,
#endif // CC_RSA_Decrypt
#if (PAD_LIST)
       (COMMAND_DESCRIPTOR_t*)0,
#endif //
#if (PAD_LIST || (CC_HMAC_Start || CC_MAC_Start))
#   if CC_HMAC_Start
        (COMMAND_DESCRIPTOR_t*)_HMAC_StartDataAddress,
#   endif
#   if CC_MAC_Start
        (COMMAND_DESCRIPTOR_t*)_MAC_StartDataAddress,
#   endif
#endif // (CC_HMAC_Start || CC_MAC_Start)
```

```
#if (PAD_LIST || CC_SequenceUpdate)
        (COMMAND_DESCRIPTOR_t*)_SequenceUpdateDataAddress,
#endif // CC_SequenceUpdate
#if (PAD_LIST || CC_Sign)
        (COMMAND_DESCRIPTOR_t*)_SignDataAddress,
#endif // CC_Sign
#if (PAD_LIST || CC_Unseal)
        (COMMAND_DESCRIPTOR_t*)_UnsealDataAddress,
#endif // CC_Unseal
#if (PAD_LIST)
       (COMMAND_DESCRIPTOR_t*)0,
#endif //
#if (PAD_LIST || CC_PolicySigned)
        (COMMAND_DESCRIPTOR_t*)_PolicySignedDataAddress,
#endif // CC_PolicySigned
#if (PAD_LIST || CC_ContextLoad)
        (COMMAND_DESCRIPTOR_t*)_ContextLoadDataAddress,
#endif // CC_ContextLoad
#if (PAD_LIST || CC_ContextSave)
        (COMMAND_DESCRIPTOR_t*)_ContextSaveDataAddress,
#endif // CC_ContextSave
#if (PAD_LIST || CC_ECDH_KeyGen)
        (COMMAND_DESCRIPTOR_t*)_ECDH_KeyGenDataAddress,
#endif // CC_ECDH_KeyGen
#if (PAD_LIST || CC_EncryptDecrypt)
        (COMMAND_DESCRIPTOR_t*)_EncryptDecryptDataAddress,
#endif // CC_EncryptDecrypt
#if (PAD_LIST || CC_FlushContext)
        (COMMAND_DESCRIPTOR_t*)_FlushContextDataAddress,
#endif // CC_FlushContext
#if (PAD_LIST)
       (COMMAND_DESCRIPTOR_t*)0,
#endif //
#if (PAD_LIST || CC_LoadExternal)
        (COMMAND_DESCRIPTOR_t*)_LoadExternalDataAddress,
#endif // CC_LoadExternal
#if (PAD_LIST || CC_MakeCredential)
        (COMMAND_DESCRIPTOR_t*)_MakeCredentialDataAddress,
#endif // CC_MakeCredential
#if (PAD_LIST || CC_NV_ReadPublic)
        (COMMAND_DESCRIPTOR_t*)_NV_ReadPublicDataAddress,
#endif // CC_NV_ReadPublic
#if (PAD_LIST || CC_PolicyAuthorize)
        (COMMAND_DESCRIPTOR_t*)_PolicyAuthorizeDataAddress,
#endif // CC_PolicyAuthorize
#if (PAD_LIST || CC_PolicyAuthValue)
        (COMMAND_DESCRIPTOR_t*)_PolicyAuthValueDataAddress,
#endif // CC_PolicyAuthValue
#if (PAD_LIST || CC_PolicyCommandCode)
        (COMMAND_DESCRIPTOR_t*)_PolicyCommandCodeDataAddress,
#endif // CC_PolicyCommandCode
#if (PAD_LIST || CC_PolicyCounterTimer)
        (COMMAND_DESCRIPTOR_t*)_PolicyCounterTimerDataAddress,
#endif // CC_PolicyCounterTimer
#if (PAD_LIST || CC_PolicyCpHash)
        (COMMAND_DESCRIPTOR_t*)_PolicyCpHashDataAddress,
#endif // CC_PolicyCpHash
#if (PAD_LIST || CC_PolicyLocality)
        (COMMAND_DESCRIPTOR_t*)_PolicyLocalityDataAddress,
#endif // CC_PolicyLocality
#if (PAD_LIST || CC_PolicyNameHash)
        (COMMAND_DESCRIPTOR_t*)_PolicyNameHashDataAddress,
#endif // CC_PolicyNameHash
#if (PAD_LIST || CC_PolicyOR)
        (COMMAND_DESCRIPTOR_t*)_PolicyORDataAddress,
#endif // CC_PolicyOR
```

```
#if (PAD_LIST || CC_PolicyTicket)
        (COMMAND_DESCRIPTOR_t*)_PolicyTicketDataAddress,
#endif // CC_PolicyTicket
#if (PAD_LIST || CC_ReadPublic)
        (COMMAND_DESCRIPTOR_t*)_ReadPublicDataAddress,
#endif // CC_ReadPublic
#if (PAD_LIST || CC_RSA_Encrypt)
        (COMMAND_DESCRIPTOR_t*)_RSA_EncryptDataAddress,
#endif // CC_RSA_Encrypt
#if (PAD_LIST)
       (COMMAND_DESCRIPTOR_t*)0,
#endif //
#if (PAD_LIST || CC_StartAuthSession)
        (COMMAND_DESCRIPTOR_t*)_StartAuthSessionDataAddress,
#endif // CC_StartAuthSession
#if (PAD_LIST || CC_VerifySignature)
        (COMMAND_DESCRIPTOR_t*)_VerifySignatureDataAddress,
#endif // CC_VerifySignature
#if (PAD_LIST || CC_ECC_Parameters)
        (COMMAND_DESCRIPTOR_t*)_ECC_ParametersDataAddress,
#endif // CC_ECC_Parameters
#if (PAD_LIST || CC_FirmwareRead)
        (COMMAND_DESCRIPTOR_t*)_FirmwareReadDataAddress,
#endif // CC_FirmwareRead
#if (PAD_LIST || CC_GetCapability)
        (COMMAND_DESCRIPTOR_t*)_GetCapabilityDataAddress,
#endif // CC_GetCapability
#if (PAD_LIST || CC_GetRandom)
        (COMMAND_DESCRIPTOR_t*)_GetRandomDataAddress,
#endif // CC_GetRandom
#if (PAD_LIST || CC_GetTestResult)
        (COMMAND_DESCRIPTOR_t*)_GetTestResultDataAddress,
#endif // CC_GetTestResult
#if (PAD_LIST || CC_Hash)
        (COMMAND_DESCRIPTOR_t*)_HashDataAddress,
#endif // CC_Hash
#if (PAD_LIST || CC_PCR_Read)
        (COMMAND_DESCRIPTOR_t*)_PCR_ReadDataAddress,
#endif // CC_PCR_Read
#if (PAD_LIST || CC_PolicyPCR)
        (COMMAND_DESCRIPTOR_t*)_PolicyPCRDataAddress,
#endif // CC_PolicyPCR
#if (PAD_LIST || CC_PolicyRestart)
        (COMMAND_DESCRIPTOR_t*)_PolicyRestartDataAddress,
#endif // CC_PolicyRestart
#if (PAD_LIST || CC_ReadClock)
        (COMMAND_DESCRIPTOR_t*)_ReadClockDataAddress,
#endif // CC_ReadClock
#if (PAD_LIST || CC_PCR_Extend)
        (COMMAND_DESCRIPTOR_t*)_PCR_ExtendDataAddress,
#endif // CC_PCR_Extend
#if (PAD_LIST || CC_PCR_SetAuthValue)
        (COMMAND_DESCRIPTOR_t*)_PCR_SetAuthValueDataAddress,
#endif // CC_PCR_SetAuthValue
#if (PAD_LIST || CC_NV_Certify)
        (COMMAND_DESCRIPTOR_t*)_NV_CertifyDataAddress,
#endif // CC_NV_Certify
#if (PAD_LIST || CC_EventSequenceComplete)
        (COMMAND_DESCRIPTOR_t*)_EventSequenceCompleteDataAddress,
#endif // CC_EventSequenceComplete
#if (PAD_LIST || CC_HashSequenceStart)
        (COMMAND_DESCRIPTOR_t*)_HashSequenceStartDataAddress,
#endif // CC_HashSequenceStart
#if (PAD_LIST || CC_PolicyPhysicalPresence)
        (COMMAND_DESCRIPTOR_t*)_PolicyPhysicalPresenceDataAddress,
#endif // CC_PolicyPhysicalPresence
```

```
#if (PAD_LIST || CC_PolicyDuplicationSelect)
        (COMMAND_DESCRIPTOR_t*)_PolicyDuplicationSelectDataAddress,
#endif // CC_PolicyDuplicationSelect
#if (PAD_LIST || CC_PolicyGetDigest)
        (COMMAND_DESCRIPTOR_t*)_PolicyGetDigestDataAddress,
#endif // CC_PolicyGetDigest
#if (PAD_LIST || CC_TestParms)
        (COMMAND_DESCRIPTOR_t*)_TestParmsDataAddress,
#endif // CC_TestParms
#if (PAD_LIST || CC_Commit)
        (COMMAND_DESCRIPTOR_t*)_CommitDataAddress,
#endif // CC_Commit
#if (PAD_LIST || CC_PolicyPassword)
        (COMMAND_DESCRIPTOR_t*)_PolicyPasswordDataAddress,
#endif // CC_PolicyPassword
#if (PAD_LIST || CC_ZGen_2Phase)
        (COMMAND_DESCRIPTOR_t*)_ZGen_2PhaseDataAddress,
#endif // CC_ZGen_2Phase
#if (PAD_LIST || CC_EC_Ephemeral)
        (COMMAND_DESCRIPTOR_t*)_EC_EphemeralDataAddress,
#endif // CC_EC_Ephemeral
#if (PAD_LIST || CC_PolicyNvWritten)
        (COMMAND_DESCRIPTOR_t*)_PolicyNvWrittenDataAddress,
#endif // CC_PolicyNvWritten
#if (PAD_LIST || CC_PolicyTemplate)
        (COMMAND_DESCRIPTOR_t*)_PolicyTemplateDataAddress,
#endif // CC_PolicyTemplate
#if (PAD_LIST || CC_CreateLoaded)
        (COMMAND_DESCRIPTOR_t*)_CreateLoadedDataAddress,
#endif // CC_CreateLoaded
#if (PAD_LIST || CC_PolicyAuthorizeNV)
        (COMMAND_DESCRIPTOR_t*)_PolicyAuthorizeNVDataAddress,
#endif // CC_PolicyAuthorizeNV
#if (PAD_LIST || CC_EncryptDecrypt2)
        (COMMAND_DESCRIPTOR_t*)_EncryptDecrypt2DataAddress,
#endif // CC_EncryptDecrypt2
#if (PAD_LIST || CC_AC_GetCapability)
        (COMMAND_DESCRIPTOR_t*)_GetCapabilityDataAddress,
#endif // CC_AC_GetCapability
#if (PAD_LIST || CC_AC_Send)
        (COMMAND_DESCRIPTOR_t*)_AC_SendDataAddress,
#endif // CC_AC_Send
#if (PAD_LIST || CC_Policy_AC_SendSelect)
        (COMMAND_DESCRIPTOR_t*)_Policy_AC_SendSelectDataAddress,
#endif // CC_Policy_AC_SendSelect
#if (PAD_LIST || CC_CertifyX509)
        (COMMAND_DESCRIPTOR_t*)_CertifyX509DataAddress,
#endif // CC_CertifyX509
#if (PAD_LIST || CC_ACT_SetTimeout)
        (COMMAND_DESCRIPTOR_t*)_ACT_SetTimeoutDataAddress,
#endif // CC_ACT_SetTimeout
#if (PAD_LIST || CC_ECC_Encrypt)
        (COMMAND_DESCRIPTOR_t*)_ECC_EncryptDataAddress,
#endif // CC_ECC_Encrypt
#if (PAD_LIST || CC_ECC_Decrypt)
        (COMMAND_DESCRIPTOR_t*)_ECC_DecryptDataAddress,
#endif // CC_ECC_Decrypt
#if (PAD_LIST || CC_PolicyCapability)
        (COMMAND_DESCRIPTOR_t*)_PolicyCapabilityDataAddress,
#endif // CC_PolicyCapability
#if (PAD_LIST || CC_PolicyParameters)
        (COMMAND_DESCRIPTOR_t*)_PolicyParametersDataAddress,
#endif // CC_PolicyParameters
#if (PAD_LIST || CC_NV_DefineSpace2)
        (COMMAND_DESCRIPTOR_t*)_NV_DefineSpace2DataAddress,
#endif // CC_NV_DefineSpace2
```

```
#if (PAD_LIST || CC_NV_ReadPublic2)
        (COMMAND_DESCRIPTOR_t*)_NV_ReadPublic2DataAddress,
#endif // CC_NV_ReadPublic2
#if (PAD_LIST || CC_SetCapability)
        (COMMAND_DESCRIPTOR_t*)_SetCapabilityDataAddress,
#endif // CC_SetCapability
#if (PAD_LIST || CC_Vendor_TCG_Test)
        (COMMAND_DESCRIPTOR_t*)_Vendor_TCG_TestDataAddress,
#endif // CC_Vendor_TCG_Test

        0
};

#endif // _COMMAND_TABLE_DISPATCH_
```

6.14 /tpm/include/private/CommandDispatcher.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

// This macro is added just so that the code is only excessively long.
#define EXIT_IF_ERROR_PLUS(x)     \
    if(TPM_RC_SUCCESS != result) \
    {                            \
        result += (x);           \
        goto Exit;               \
    }
#if CC_Startup
case TPM_CC_Startup:
{
    Startup_In* in = (Startup_In*)MemoryGetInBuffer(sizeof(Startup_In));
    result         = TPM_SU_Unmarshal(&in->startupType, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Startup_startupType);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Startup(in);
    break;
}
#endif  // CC_Startup
#if CC_Shutdown
case TPM_CC_Shutdown:
{
    Shutdown_In* in = (Shutdown_In*)MemoryGetInBuffer(sizeof(Shutdown_In));
    result = TPM_SU_Unmarshal(&in->shutdownType, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Shutdown_shutdownType);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Shutdown(in);
    break;
}
#endif  // CC_Shutdown
#if CC_SelfTest
case TPM_CC_SelfTest:
{
    SelfTest_In* in = (SelfTest_In*)MemoryGetInBuffer(sizeof(SelfTest_In));
    result = TPMI_YES_NO_Unmarshal(&in->fullTest, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_SelfTest_fullTest);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
```

```
        goto Exit;
    }
    result = TPM2_SelfTest(in);
    break;
}
#endif  // CC_SelfTest
#if CC_IncrementalSelfTest
case TPM_CC_IncrementalSelfTest:
{
    IncrementalSelfTest_In* in =
        (IncrementalSelfTest_In*)MemoryGetInBuffer(sizeof(IncrementalSelfTest_In));
    IncrementalSelfTest_Out* out =
        (IncrementalSelfTest_Out*)MemoryGetOutBuffer(sizeof(IncrementalSelfTest_Out));
    result = TPML_ALG_Unmarshal(&in->toTest, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_IncrementalSelfTest_toTest);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_IncrementalSelfTest(in, out);
    rSize  = sizeof(IncrementalSelfTest_Out);
    *respParmSize += TPML_ALG_Marshal(&out->toDoList, responseBuffer, &rSize);
    break;
}
#endif  // CC_IncrementalSelfTest
#if CC_GetTestResult
case TPM_CC_GetTestResult:
{
    GetTestResult_Out* out =
        (GetTestResult_Out*)MemoryGetOutBuffer(sizeof(GetTestResult_Out));
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_GetTestResult(out);
    rSize  = sizeof(GetTestResult_Out);
    *respParmSize += TPM2B_MAX_BUFFER_Marshal(&out->outData, responseBuffer, &rSize);
    *respParmSize += TPM_RC_Marshal(&out->testResult, responseBuffer, &rSize);
    break;
}
#endif  // CC_GetTestResult
#if CC_StartAuthSession
case TPM_CC_StartAuthSession:
{
    StartAuthSession_In* in =
        (StartAuthSession_In*)MemoryGetInBuffer(sizeof(StartAuthSession_In));
    StartAuthSession_Out* out =
        (StartAuthSession_Out*)MemoryGetOutBuffer(sizeof(StartAuthSession_Out));
    in->tpmKey = handles[0];
    in->bind   = handles[1];
    result = TPM2B_NONCE_Unmarshal(&in->nonceCaller, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_StartAuthSession_nonceCaller);
    result = TPM2B_ENCRYPTED_SECRET_Unmarshal(
        &in->encryptedSalt, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_StartAuthSession_encryptedSalt);
    result = TPM_SE_Unmarshal(&in->sessionType, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_StartAuthSession_sessionType);
    result =
        TPMT_SYM_DEF_Unmarshal(&in->symmetric, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_StartAuthSession_symmetric);
    result =
        TPMI_ALG_HASH_Unmarshal(&in->authHash, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_StartAuthSession_authHash);
    if(*paramBufferSize != 0)
```

```
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_StartAuthSession(in, out);
    rSize  = sizeof(StartAuthSession_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->sessionHandle;
    *respParmSize += TPM2B_NONCE_Marshal(&out->nonceTPM, responseBuffer, &rSize);
    break;
}
#endif  // CC_StartAuthSession
#if CC_PolicyRestart
case TPM_CC_PolicyRestart:
{
    PolicyRestart_In* in =
        (PolicyRestart_In*)MemoryGetInBuffer(sizeof(PolicyRestart_In));
    in->sessionHandle = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyRestart(in);
    break;
}
#endif  // CC_PolicyRestart
#if CC_Create
case TPM_CC_Create:
{
    Create_In*  in   = (Create_In*)MemoryGetInBuffer(sizeof(Create_In));
    Create_Out* out  = (Create_Out*)MemoryGetOutBuffer(sizeof(Create_Out));
    in->parentHandle = handles[0];
    result           = TPM2B_SENSITIVE_CREATE_Unmarshal(
        &in->inSensitive, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Create_inSensitive);
    result =
        TPM2B_PUBLIC_Unmarshal(&in->inPublic, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_Create_inPublic);
    result = TPM2B_DATA_Unmarshal(&in->outsideInfo, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Create_outsideInfo);
    result =
        TPML_PCR_SELECTION_Unmarshal(&in->creationPCR, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Create_creationPCR);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Create(in, out);
    rSize  = sizeof(Create_Out);
    *respParmSize += TPM2B_PRIVATE_Marshal(&out->outPrivate, responseBuffer, &rSize);
    *respParmSize += TPM2B_PUBLIC_Marshal(&out->outPublic, responseBuffer, &rSize);
    *respParmSize +=
        TPM2B_CREATION_DATA_Marshal(&out->creationData, responseBuffer, &rSize);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->creationHash, responseBuffer, &rSize);
    *respParmSize +=
        TPMT_TK_CREATION_Marshal(&out->creationTicket, responseBuffer, &rSize);
    break;
}
#endif  // CC_Create
#if CC_Load
case TPM_CC_Load:
{
    Load_In*  in     = (Load_In*)MemoryGetInBuffer(sizeof(Load_In));
```

```
    Load_Out* out    = (Load_Out*)MemoryGetOutBuffer(sizeof(Load_Out));
    in->parentHandle = handles[0];
    result = TPM2B_PRIVATE_Unmarshal(&in->inPrivate, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Load_inPrivate);
    result =
        TPM2B_PUBLIC_Unmarshal(&in->inPublic, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_Load_inPublic);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Load(in, out);
    rSize  = sizeof(Load_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->objectHandle;
    *respParmSize += TPM2B_NAME_Marshal(&out->name, responseBuffer, &rSize);
    break;
}
#endif  // CC_Load
#if CC_LoadExternal
case TPM_CC_LoadExternal:
{
    LoadExternal_In* in =
        (LoadExternal_In*)MemoryGetInBuffer(sizeof(LoadExternal_In));
    LoadExternal_Out* out =
        (LoadExternal_Out*)MemoryGetOutBuffer(sizeof(LoadExternal_Out));
    result = TPM2B_SENSITIVE_Unmarshal(&in->inPrivate, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_LoadExternal_inPrivate);
    result =
        TPM2B_PUBLIC_Unmarshal(&in->inPublic, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_LoadExternal_inPublic);
    result = TPMI_RH_HIERARCHY_Unmarshal(
        &in->hierarchy, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_LoadExternal_hierarchy);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_LoadExternal(in, out);
    rSize  = sizeof(LoadExternal_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->objectHandle;
    *respParmSize += TPM2B_NAME_Marshal(&out->name, responseBuffer, &rSize);
    break;
}
#endif  // CC_LoadExternal
#if CC_ReadPublic
case TPM_CC_ReadPublic:
{
    ReadPublic_In*  in  = (ReadPublic_In*)MemoryGetInBuffer(sizeof(ReadPublic_In));
    ReadPublic_Out* out = (ReadPublic_Out*)MemoryGetOutBuffer(sizeof(ReadPublic_Out));
    in->objectHandle    = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ReadPublic(in, out);
    rSize  = sizeof(ReadPublic_Out);
    *respParmSize += TPM2B_PUBLIC_Marshal(&out->outPublic, responseBuffer, &rSize);
    *respParmSize += TPM2B_NAME_Marshal(&out->name, responseBuffer, &rSize);
    *respParmSize += TPM2B_NAME_Marshal(&out->qualifiedName, responseBuffer, &rSize);
```

```
    break;
}
#endif  // CC_ReadPublic
#if CC_ActivateCredential
case TPM_CC_ActivateCredential:
{
    ActivateCredential_In* in =
        (ActivateCredential_In*)MemoryGetInBuffer(sizeof(ActivateCredential_In));
    ActivateCredential_Out* out =
        (ActivateCredential_Out*)MemoryGetOutBuffer(sizeof(ActivateCredential_Out));
    in->activateHandle = handles[0];
    in->keyHandle      = handles[1];
    result =
        TPM2B_ID_OBJECT_Unmarshal(&in->credentialBlob, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ActivateCredential_credentialBlob);
    result =
        TPM2B_ENCRYPTED_SECRET_Unmarshal(&in->secret, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ActivateCredential_secret);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ActivateCredential(in, out);
    rSize  = sizeof(ActivateCredential_Out);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->certInfo, responseBuffer, &rSize);
    break;
}
#endif  // CC_ActivateCredential
#if CC_MakeCredential
case TPM_CC_MakeCredential:
{
    MakeCredential_In* in =
        (MakeCredential_In*)MemoryGetInBuffer(sizeof(MakeCredential_In));
    MakeCredential_Out* out =
        (MakeCredential_Out*)MemoryGetOutBuffer(sizeof(MakeCredential_Out));
    in->handle = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->credential, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_MakeCredential_credential);
    result = TPM2B_NAME_Unmarshal(&in->objectName, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_MakeCredential_objectName);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_MakeCredential(in, out);
    rSize  = sizeof(MakeCredential_Out);
    *respParmSize +=
        TPM2B_ID_OBJECT_Marshal(&out->credentialBlob, responseBuffer, &rSize);
    *respParmSize +=
        TPM2B_ENCRYPTED_SECRET_Marshal(&out->secret, responseBuffer, &rSize);
    break;
}
#endif  // CC_MakeCredential
#if CC_Unseal
case TPM_CC_Unseal:
{
    Unseal_In*  in  = (Unseal_In*)MemoryGetInBuffer(sizeof(Unseal_In));
    Unseal_Out* out = (Unseal_Out*)MemoryGetOutBuffer(sizeof(Unseal_Out));
    in->itemHandle  = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
```

```
    result = TPM2_Unseal(in, out);
    rSize  = sizeof(Unseal_Out);
    *respParmSize +=
        TPM2B_SENSITIVE_DATA_Marshal(&out->outData, responseBuffer, &rSize);
    break;
}
#endif  // CC_Unseal
#if CC_ObjectChangeAuth
case TPM_CC_ObjectChangeAuth:
{
    ObjectChangeAuth_In* in =
        (ObjectChangeAuth_In*)MemoryGetInBuffer(sizeof(ObjectChangeAuth_In));
    ObjectChangeAuth_Out* out =
        (ObjectChangeAuth_Out*)MemoryGetOutBuffer(sizeof(ObjectChangeAuth_Out));
    in->objectHandle = handles[0];
    in->parentHandle = handles[1];
    result = TPM2B_AUTH_Unmarshal(&in->newAuth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ObjectChangeAuth_newAuth);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ObjectChangeAuth(in, out);
    rSize  = sizeof(ObjectChangeAuth_Out);
    *respParmSize += TPM2B_PRIVATE_Marshal(&out->outPrivate, responseBuffer, &rSize);
    break;
}
#endif  // CC_ObjectChangeAuth
#if CC_CreateLoaded
case TPM_CC_CreateLoaded:
{
    CreateLoaded_In* in =
        (CreateLoaded_In*)MemoryGetInBuffer(sizeof(CreateLoaded_In));
    CreateLoaded_Out* out =
        (CreateLoaded_Out*)MemoryGetOutBuffer(sizeof(CreateLoaded_Out));
    in->parentHandle = handles[0];
    result           = TPM2B_SENSITIVE_CREATE_Unmarshal(
        &in->inSensitive, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CreateLoaded_inSensitive);
    result = TPM2B_TEMPLATE_Unmarshal(&in->inPublic, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CreateLoaded_inPublic);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_CreateLoaded(in, out);
    rSize  = sizeof(CreateLoaded_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->objectHandle;
    *respParmSize += TPM2B_PRIVATE_Marshal(&out->outPrivate, responseBuffer, &rSize);
    *respParmSize += TPM2B_PUBLIC_Marshal(&out->outPublic, responseBuffer, &rSize);
    *respParmSize += TPM2B_NAME_Marshal(&out->name, responseBuffer, &rSize);
    break;
}
#endif  // CC_CreateLoaded
#if CC_Duplicate
case TPM_CC_Duplicate:
{
    Duplicate_In*  in   = (Duplicate_In*)MemoryGetInBuffer(sizeof(Duplicate_In));
    Duplicate_Out* out  = (Duplicate_Out*)MemoryGetOutBuffer(sizeof(Duplicate_Out));
    in->objectHandle    = handles[0];
    in->newParentHandle = handles[1];
    result = TPM2B_DATA_Unmarshal(&in->encryptionKeyIn, paramBuffer, paramBufferSize);
```

```
    EXIT_IF_ERROR_PLUS(RC_Duplicate_encryptionKeyIn);
    result = TPMT_SYM_DEF_OBJECT_Unmarshal(
        &in->symmetricAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_Duplicate_symmetricAlg);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Duplicate(in, out);
    rSize  = sizeof(Duplicate_Out);
    *respParmSize +=
        TPM2B_DATA_Marshal(&out->encryptionKeyOut, responseBuffer, &rSize);
    *respParmSize += TPM2B_PRIVATE_Marshal(&out->duplicate, responseBuffer, &rSize);
    *respParmSize +=
        TPM2B_ENCRYPTED_SECRET_Marshal(&out->outSymSeed, responseBuffer, &rSize);
    break;
}
#endif  // CC_Duplicate
#if CC_Rewrap
case TPM_CC_Rewrap:
{
    Rewrap_In*  in  = (Rewrap_In*)MemoryGetInBuffer(sizeof(Rewrap_In));
    Rewrap_Out* out = (Rewrap_Out*)MemoryGetOutBuffer(sizeof(Rewrap_Out));
    in->oldParent   = handles[0];
    in->newParent   = handles[1];
    result = TPM2B_PRIVATE_Unmarshal(&in->inDuplicate, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Rewrap_inDuplicate);
    result = TPM2B_NAME_Unmarshal(&in->name, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Rewrap_name);
    result = TPM2B_ENCRYPTED_SECRET_Unmarshal(
        &in->inSymSeed, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Rewrap_inSymSeed);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Rewrap(in, out);
    rSize  = sizeof(Rewrap_Out);
    *respParmSize +=
        TPM2B_PRIVATE_Marshal(&out->outDuplicate, responseBuffer, &rSize);
    *respParmSize +=
        TPM2B_ENCRYPTED_SECRET_Marshal(&out->outSymSeed, responseBuffer, &rSize);
    break;
}
#endif  // CC_Rewrap
#if CC_Import
case TPM_CC_Import:
{
    Import_In*  in   = (Import_In*)MemoryGetInBuffer(sizeof(Import_In));
    Import_Out* out  = (Import_Out*)MemoryGetOutBuffer(sizeof(Import_Out));
    in->parentHandle = handles[0];
    result = TPM2B_DATA_Unmarshal(&in->encryptionKey, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Import_encryptionKey);
    result = TPM2B_PUBLIC_Unmarshal(
        &in->objectPublic, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_Import_objectPublic);
    result = TPM2B_PRIVATE_Unmarshal(&in->duplicate, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Import_duplicate);
    result = TPM2B_ENCRYPTED_SECRET_Unmarshal(
        &in->inSymSeed, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Import_inSymSeed);
    result = TPMT_SYM_DEF_OBJECT_Unmarshal(
        &in->symmetricAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_Import_symmetricAlg);
```

```
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Import(in, out);
    rSize  = sizeof(Import_Out);
    *respParmSize += TPM2B_PRIVATE_Marshal(&out->outPrivate, responseBuffer, &rSize);
    break;
}
#endif  // CC_Import
#if CC_RSA_Encrypt
case TPM_CC_RSA_Encrypt:
{
    RSA_Encrypt_In*  in = (RSA_Encrypt_In*)MemoryGetInBuffer(sizeof(RSA_Encrypt_In));
    RSA_Encrypt_Out* out =
        (RSA_Encrypt_Out*)MemoryGetOutBuffer(sizeof(RSA_Encrypt_Out));
    in->keyHandle = handles[0];
    result =
        TPM2B_PUBLIC_KEY_RSA_Unmarshal(&in->message, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_RSA_Encrypt_message);
    result =
        TPMT_RSA_DECRYPT_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_RSA_Encrypt_inScheme);
    result = TPM2B_DATA_Unmarshal(&in->label, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_RSA_Encrypt_label);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_RSA_Encrypt(in, out);
    rSize  = sizeof(RSA_Encrypt_Out);
    *respParmSize +=
        TPM2B_PUBLIC_KEY_RSA_Marshal(&out->outData, responseBuffer, &rSize);
    break;
}
#endif  // CC_RSA_Encrypt
#if CC_RSA_Decrypt
case TPM_CC_RSA_Decrypt:
{
    RSA_Decrypt_In*  in = (RSA_Decrypt_In*)MemoryGetInBuffer(sizeof(RSA_Decrypt_In));
    RSA_Decrypt_Out* out =
        (RSA_Decrypt_Out*)MemoryGetOutBuffer(sizeof(RSA_Decrypt_Out));
    in->keyHandle = handles[0];
    result =
        TPM2B_PUBLIC_KEY_RSA_Unmarshal(&in->cipherText, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_RSA_Decrypt_cipherText);
    result =
        TPMT_RSA_DECRYPT_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_RSA_Decrypt_inScheme);
    result = TPM2B_DATA_Unmarshal(&in->label, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_RSA_Decrypt_label);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_RSA_Decrypt(in, out);
    rSize  = sizeof(RSA_Decrypt_Out);
    *respParmSize +=
        TPM2B_PUBLIC_KEY_RSA_Marshal(&out->message, responseBuffer, &rSize);
    break;
}
#endif  // CC_RSA_Decrypt
#if CC_ECDH_KeyGen
```

```
case TPM_CC_ECDH_KeyGen:
{
    ECDH_KeyGen_In*  in = (ECDH_KeyGen_In*)MemoryGetInBuffer(sizeof(ECDH_KeyGen_In));
    ECDH_KeyGen_Out* out =
        (ECDH_KeyGen_Out*)MemoryGetOutBuffer(sizeof(ECDH_KeyGen_Out));
    in->keyHandle = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ECDH_KeyGen(in, out);
    rSize  = sizeof(ECDH_KeyGen_Out);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->zPoint, responseBuffer, &rSize);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->pubPoint, responseBuffer, &rSize);
    break;
}
#endif  // CC_ECDH_KeyGen
#if CC_ECDH_ZGen
case TPM_CC_ECDH_ZGen:
{
    ECDH_ZGen_In*  in  = (ECDH_ZGen_In*)MemoryGetInBuffer(sizeof(ECDH_ZGen_In));
    ECDH_ZGen_Out* out = (ECDH_ZGen_Out*)MemoryGetOutBuffer(sizeof(ECDH_ZGen_Out));
    in->keyHandle      = handles[0];
    result = TPM2B_ECC_POINT_Unmarshal(&in->inPoint, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ECDH_ZGen_inPoint);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ECDH_ZGen(in, out);
    rSize  = sizeof(ECDH_ZGen_Out);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->outPoint, responseBuffer, &rSize);
    break;
}
#endif  // CC_ECDH_ZGen
#if CC_ECC_Parameters
case TPM_CC_ECC_Parameters:
{
    ECC_Parameters_In* in =
        (ECC_Parameters_In*)MemoryGetInBuffer(sizeof(ECC_Parameters_In));
    ECC_Parameters_Out* out =
        (ECC_Parameters_Out*)MemoryGetOutBuffer(sizeof(ECC_Parameters_Out));
    result =
        TPMI_ECC_CURVE_Unmarshal(&in->curveID, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_ECC_Parameters_curveID);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ECC_Parameters(in, out);
    rSize  = sizeof(ECC_Parameters_Out);
    *respParmSize +=
        TPMS_ALGORITHM_DETAIL_ECC_Marshal(&out->parameters, responseBuffer, &rSize);
    break;
}
#endif  // CC_ECC_Parameters
#if CC_ZGen_2Phase
case TPM_CC_ZGen_2Phase:
{
    ZGen_2Phase_In*  in = (ZGen_2Phase_In*)MemoryGetInBuffer(sizeof(ZGen_2Phase_In));
    ZGen_2Phase_Out* out =
        (ZGen_2Phase_Out*)MemoryGetOutBuffer(sizeof(ZGen_2Phase_Out));
    in->keyA = handles[0];
```

```
    result   = TPM2B_ECC_POINT_Unmarshal(&in->inQsB, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ZGen_2Phase_inQsB);
    result = TPM2B_ECC_POINT_Unmarshal(&in->inQeB, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ZGen_2Phase_inQeB);
    result = TPMI_ECC_KEY_EXCHANGE_Unmarshal(
        &in->inScheme, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_ZGen_2Phase_inScheme);
    result = UINT16_Unmarshal(&in->counter, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ZGen_2Phase_counter);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ZGen_2Phase(in, out);
    rSize  = sizeof(ZGen_2Phase_Out);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->outZ1, responseBuffer, &rSize);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->outZ2, responseBuffer, &rSize);
    break;
}
#endif  // CC_ZGen_2Phase
#if CC_ECC_Encrypt
case TPM_CC_ECC_Encrypt:
{
    ECC_Encrypt_In*  in = (ECC_Encrypt_In*)MemoryGetInBuffer(sizeof(ECC_Encrypt_In));
    ECC_Encrypt_Out* out =
        (ECC_Encrypt_Out*)MemoryGetOutBuffer(sizeof(ECC_Encrypt_Out));
    in->keyHandle = handles[0];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->plainText, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ECC_Encrypt_plainText);
    result =
        TPMT_KDF_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_ECC_Encrypt_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ECC_Encrypt(in, out);
    rSize  = sizeof(ECC_Encrypt_Out);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->C1, responseBuffer, &rSize);
    *respParmSize += TPM2B_MAX_BUFFER_Marshal(&out->C2, responseBuffer, &rSize);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->C3, responseBuffer, &rSize);
    break;
}
#endif  // CC_ECC_Encrypt
#if CC_ECC_Decrypt
case TPM_CC_ECC_Decrypt:
{
    ECC_Decrypt_In*  in = (ECC_Decrypt_In*)MemoryGetInBuffer(sizeof(ECC_Decrypt_In));
    ECC_Decrypt_Out* out =
        (ECC_Decrypt_Out*)MemoryGetOutBuffer(sizeof(ECC_Decrypt_Out));
    in->keyHandle = handles[0];
    result        = TPM2B_ECC_POINT_Unmarshal(&in->C1, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ECC_Decrypt_C1);
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->C2, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ECC_Decrypt_C2);
    result = TPM2B_DIGEST_Unmarshal(&in->C3, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ECC_Decrypt_C3);
    result =
        TPMT_KDF_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_ECC_Decrypt_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
```

```
    }
    result = TPM2_ECC_Decrypt(in, out);
    rSize  = sizeof(ECC_Decrypt_Out);
    *respParmSize +=
        TPM2B_MAX_BUFFER_Marshal(&out->plainText, responseBuffer, &rSize);
    break;
}
#endif  // CC_ECC_Decrypt
#if CC_EncryptDecrypt
case TPM_CC_EncryptDecrypt:
{
    EncryptDecrypt_In* in =
        (EncryptDecrypt_In*)MemoryGetInBuffer(sizeof(EncryptDecrypt_In));
    EncryptDecrypt_Out* out =
        (EncryptDecrypt_Out*)MemoryGetOutBuffer(sizeof(EncryptDecrypt_Out));
    in->keyHandle = handles[0];
    result        = TPMI_YES_NO_Unmarshal(&in->decrypt, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt_decrypt);
    result =
        TPMI_ALG_CIPHER_MODE_Unmarshal(&in->mode, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt_mode);
    result = TPM2B_IV_Unmarshal(&in->ivIn, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt_ivIn);
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->inData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt_inData);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_EncryptDecrypt(in, out);
    rSize  = sizeof(EncryptDecrypt_Out);
    *respParmSize += TPM2B_MAX_BUFFER_Marshal(&out->outData, responseBuffer, &rSize);
    *respParmSize += TPM2B_IV_Marshal(&out->ivOut, responseBuffer, &rSize);
    break;
}
#endif  // CC_EncryptDecrypt
#if CC_EncryptDecrypt2
case TPM_CC_EncryptDecrypt2:
{
    EncryptDecrypt2_In* in =
        (EncryptDecrypt2_In*)MemoryGetInBuffer(sizeof(EncryptDecrypt2_In));
    EncryptDecrypt2_Out* out =
        (EncryptDecrypt2_Out*)MemoryGetOutBuffer(sizeof(EncryptDecrypt2_Out));
    in->keyHandle = handles[0];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->inData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt2_inData);
    result = TPMI_YES_NO_Unmarshal(&in->decrypt, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt2_decrypt);
    result =
        TPMI_ALG_CIPHER_MODE_Unmarshal(&in->mode, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt2_mode);
    result = TPM2B_IV_Unmarshal(&in->ivIn, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EncryptDecrypt2_ivIn);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_EncryptDecrypt2(in, out);
    rSize  = sizeof(EncryptDecrypt2_Out);
    *respParmSize += TPM2B_MAX_BUFFER_Marshal(&out->outData, responseBuffer, &rSize);
    *respParmSize += TPM2B_IV_Marshal(&out->ivOut, responseBuffer, &rSize);
    break;
}
#endif  // CC_EncryptDecrypt2
```

```
#if CC_Hash
case TPM_CC_Hash:
{
    Hash_In*  in  = (Hash_In*)MemoryGetInBuffer(sizeof(Hash_In));
    Hash_Out* out = (Hash_Out*)MemoryGetOutBuffer(sizeof(Hash_Out));
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->data, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Hash_data);
    result =
        TPMI_ALG_HASH_Unmarshal(&in->hashAlg, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_Hash_hashAlg);
    result = TPMI_RH_HIERARCHY_Unmarshal(
        &in->hierarchy, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_Hash_hierarchy);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Hash(in, out);
    rSize  = sizeof(Hash_Out);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->outHash, responseBuffer, &rSize);
    *respParmSize +=
        TPMT_TK_HASHCHECK_Marshal(&out->validation, responseBuffer, &rSize);
    break;
}
#endif  // CC_Hash
#if CC_HMAC
case TPM_CC_HMAC:
{
    HMAC_In*  in  = (HMAC_In*)MemoryGetInBuffer(sizeof(HMAC_In));
    HMAC_Out* out = (HMAC_Out*)MemoryGetOutBuffer(sizeof(HMAC_Out));
    in->handle    = handles[0];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->buffer, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_HMAC_buffer);
    result =
        TPMI_ALG_HASH_Unmarshal(&in->hashAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_HMAC_hashAlg);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_HMAC(in, out);
    rSize  = sizeof(HMAC_Out);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->outHMAC, responseBuffer, &rSize);
    break;
}
#endif  // CC_HMAC
#if CC_MAC
case TPM_CC_MAC:
{
    MAC_In*  in  = (MAC_In*)MemoryGetInBuffer(sizeof(MAC_In));
    MAC_Out* out = (MAC_Out*)MemoryGetOutBuffer(sizeof(MAC_Out));
    in->handle   = handles[0];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->buffer, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_MAC_buffer);
    result = TPMI_ALG_MAC_SCHEME_Unmarshal(
        &in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_MAC_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_MAC(in, out);
    rSize  = sizeof(MAC_Out);
```

```
    *respParmSize += TPM2B_DIGEST_Marshal(&out->outMAC, responseBuffer, &rSize);
    break;
}
#endif  // CC_MAC
#if CC_GetRandom
case TPM_CC_GetRandom:
{
    GetRandom_In*  in  = (GetRandom_In*)MemoryGetInBuffer(sizeof(GetRandom_In));
    GetRandom_Out* out = (GetRandom_Out*)MemoryGetOutBuffer(sizeof(GetRandom_Out));
    result = UINT16_Unmarshal(&in->bytesRequested, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_GetRandom_bytesRequested);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_GetRandom(in, out);
    rSize  = sizeof(GetRandom_Out);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->randomBytes, responseBuffer, &rSize);
    break;
}
#endif  // CC_GetRandom
#if CC_StirRandom
case TPM_CC_StirRandom:
{
    StirRandom_In* in = (StirRandom_In*)MemoryGetInBuffer(sizeof(StirRandom_In));
    result =
        TPM2B_SENSITIVE_DATA_Unmarshal(&in->inData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_StirRandom_inData);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_StirRandom(in);
    break;
}
#endif  // CC_StirRandom
#if CC_HMAC_Start
case TPM_CC_HMAC_Start:
{
    HMAC_Start_In*  in  = (HMAC_Start_In*)MemoryGetInBuffer(sizeof(HMAC_Start_In));
    HMAC_Start_Out* out = (HMAC_Start_Out*)MemoryGetOutBuffer(sizeof(HMAC_Start_Out));
    in->handle          = handles[0];
    result = TPM2B_AUTH_Unmarshal(&in->auth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_HMAC_Start_auth);
    result =
        TPMI_ALG_HASH_Unmarshal(&in->hashAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_HMAC_Start_hashAlg);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_HMAC_Start(in, out);
    rSize  = sizeof(HMAC_Start_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->sequenceHandle;
    break;
}
#endif  // CC_HMAC_Start
#if CC_MAC_Start
case TPM_CC_MAC_Start:
{
    MAC_Start_In*  in  = (MAC_Start_In*)MemoryGetInBuffer(sizeof(MAC_Start_In));
```

```
    MAC_Start_Out* out = (MAC_Start_Out*)MemoryGetOutBuffer(sizeof(MAC_Start_Out));
    in->handle         = handles[0];
    result = TPM2B_AUTH_Unmarshal(&in->auth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_MAC_Start_auth);
    result = TPMI_ALG_MAC_SCHEME_Unmarshal(
        &in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_MAC_Start_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_MAC_Start(in, out);
    rSize  = sizeof(MAC_Start_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->sequenceHandle;
    break;
}
#endif  // CC_MAC_Start
#if CC_HashSequenceStart
case TPM_CC_HashSequenceStart:
{
    HashSequenceStart_In* in =
        (HashSequenceStart_In*)MemoryGetInBuffer(sizeof(HashSequenceStart_In));
    HashSequenceStart_Out* out =
        (HashSequenceStart_Out*)MemoryGetOutBuffer(sizeof(HashSequenceStart_Out));
    result = TPM2B_AUTH_Unmarshal(&in->auth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_HashSequenceStart_auth);
    result =
        TPMI_ALG_HASH_Unmarshal(&in->hashAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_HashSequenceStart_hashAlg);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_HashSequenceStart(in, out);
    rSize  = sizeof(HashSequenceStart_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->sequenceHandle;
    break;
}
#endif  // CC_HashSequenceStart
#if CC_SequenceUpdate
case TPM_CC_SequenceUpdate:
{
    SequenceUpdate_In* in =
        (SequenceUpdate_In*)MemoryGetInBuffer(sizeof(SequenceUpdate_In));
    in->sequenceHandle = handles[0];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->buffer, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_SequenceUpdate_buffer);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_SequenceUpdate(in);
    break;
}
#endif  // CC_SequenceUpdate
#if CC_SequenceComplete
case TPM_CC_SequenceComplete:
{
    SequenceComplete_In* in =
```

```
        (SequenceComplete_In*)MemoryGetInBuffer(sizeof(SequenceComplete_In));
    SequenceComplete_Out* out =
        (SequenceComplete_Out*)MemoryGetOutBuffer(sizeof(SequenceComplete_Out));
    in->sequenceHandle = handles[0];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->buffer, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_SequenceComplete_buffer);
    result = TPMI_RH_HIERARCHY_Unmarshal(
        &in->hierarchy, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_SequenceComplete_hierarchy);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_SequenceComplete(in, out);
    rSize  = sizeof(SequenceComplete_Out);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->result, responseBuffer, &rSize);
    *respParmSize +=
        TPMT_TK_HASHCHECK_Marshal(&out->validation, responseBuffer, &rSize);
    break;
}
#endif  // CC_SequenceComplete
#if CC_EventSequenceComplete
case TPM_CC_EventSequenceComplete:
{
    EventSequenceComplete_In* in = (EventSequenceComplete_In*)MemoryGetInBuffer(
        sizeof(EventSequenceComplete_In));
    EventSequenceComplete_Out* out = (EventSequenceComplete_Out*)MemoryGetOutBuffer(
        sizeof(EventSequenceComplete_Out));
    in->pcrHandle      = handles[0];
    in->sequenceHandle = handles[1];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->buffer, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EventSequenceComplete_buffer);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_EventSequenceComplete(in, out);
    rSize  = sizeof(EventSequenceComplete_Out);
    *respParmSize +=
        TPML_DIGEST_VALUES_Marshal(&out->results, responseBuffer, &rSize);
    break;
}
#endif  // CC_EventSequenceComplete
#if CC_Certify
case TPM_CC_Certify:
{
    Certify_In*  in  = (Certify_In*)MemoryGetInBuffer(sizeof(Certify_In));
    Certify_Out* out = (Certify_Out*)MemoryGetOutBuffer(sizeof(Certify_Out));
    in->objectHandle = handles[0];
    in->signHandle   = handles[1];
    result = TPM2B_DATA_Unmarshal(&in->qualifyingData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Certify_qualifyingData);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_Certify_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Certify(in, out);
    rSize  = sizeof(Certify_Out);
    *respParmSize += TPM2B_ATTEST_Marshal(&out->certifyInfo, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
```

```
    break;
}
#endif  // CC_Certify
#if CC_CertifyCreation
case TPM_CC_CertifyCreation:
{
    CertifyCreation_In* in =
        (CertifyCreation_In*)MemoryGetInBuffer(sizeof(CertifyCreation_In));
    CertifyCreation_Out* out =
        (CertifyCreation_Out*)MemoryGetOutBuffer(sizeof(CertifyCreation_Out));
    in->signHandle   = handles[0];
    in->objectHandle = handles[1];
    result = TPM2B_DATA_Unmarshal(&in->qualifyingData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CertifyCreation_qualifyingData);
    result = TPM2B_DIGEST_Unmarshal(&in->creationHash, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CertifyCreation_creationHash);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_CertifyCreation_inScheme);
    result =
        TPMT_TK_CREATION_Unmarshal(&in->creationTicket, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CertifyCreation_creationTicket);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_CertifyCreation(in, out);
    rSize  = sizeof(CertifyCreation_Out);
    *respParmSize += TPM2B_ATTEST_Marshal(&out->certifyInfo, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
#endif  // CC_CertifyCreation
#if CC_Quote
case TPM_CC_Quote:
{
    Quote_In*  in  = (Quote_In*)MemoryGetInBuffer(sizeof(Quote_In));
    Quote_Out* out = (Quote_Out*)MemoryGetOutBuffer(sizeof(Quote_Out));
    in->signHandle = handles[0];
    result = TPM2B_DATA_Unmarshal(&in->qualifyingData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Quote_qualifyingData);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_Quote_inScheme);
    result =
        TPML_PCR_SELECTION_Unmarshal(&in->PCRselect, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Quote_PCRselect);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Quote(in, out);
    rSize  = sizeof(Quote_Out);
    *respParmSize += TPM2B_ATTEST_Marshal(&out->quoted, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
#endif  // CC_Quote
#if CC_GetSessionAuditDigest
case TPM_CC_GetSessionAuditDigest:
{
    GetSessionAuditDigest_In* in = (GetSessionAuditDigest_In*)MemoryGetInBuffer(
        sizeof(GetSessionAuditDigest_In));
    GetSessionAuditDigest_Out* out = (GetSessionAuditDigest_Out*)MemoryGetOutBuffer(
```

```
        sizeof(GetSessionAuditDigest_Out));
    in->privacyAdminHandle = handles[0];
    in->signHandle         = handles[1];
    in->sessionHandle      = handles[2];
    result = TPM2B_DATA_Unmarshal(&in->qualifyingData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_GetSessionAuditDigest_qualifyingData);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_GetSessionAuditDigest_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_GetSessionAuditDigest(in, out);
    rSize  = sizeof(GetSessionAuditDigest_Out);
    *respParmSize += TPM2B_ATTEST_Marshal(&out->auditInfo, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
#endif  // CC_GetSessionAuditDigest
#if CC_GetCommandAuditDigest
case TPM_CC_GetCommandAuditDigest:
{
    GetCommandAuditDigest_In* in = (GetCommandAuditDigest_In*)MemoryGetInBuffer(
        sizeof(GetCommandAuditDigest_In));
    GetCommandAuditDigest_Out* out = (GetCommandAuditDigest_Out*)MemoryGetOutBuffer(
        sizeof(GetCommandAuditDigest_Out));
    in->privacyHandle = handles[0];
    in->signHandle    = handles[1];
    result = TPM2B_DATA_Unmarshal(&in->qualifyingData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_GetCommandAuditDigest_qualifyingData);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_GetCommandAuditDigest_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_GetCommandAuditDigest(in, out);
    rSize  = sizeof(GetCommandAuditDigest_Out);
    *respParmSize += TPM2B_ATTEST_Marshal(&out->auditInfo, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
#endif  // CC_GetCommandAuditDigest
#if CC_GetTime
case TPM_CC_GetTime:
{
    GetTime_In*  in        = (GetTime_In*)MemoryGetInBuffer(sizeof(GetTime_In));
    GetTime_Out* out       = (GetTime_Out*)MemoryGetOutBuffer(sizeof(GetTime_Out));
    in->privacyAdminHandle = handles[0];
    in->signHandle         = handles[1];
    result = TPM2B_DATA_Unmarshal(&in->qualifyingData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_GetTime_qualifyingData);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_GetTime_inScheme);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_GetTime(in, out);
    rSize  = sizeof(GetTime_Out);
```

```
    *respParmSize += TPM2B_ATTEST_Marshal(&out->timeInfo, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
#endif  // CC_GetTime
#if CC_CertifyX509
case TPM_CC_CertifyX509:
{
    CertifyX509_In*  in = (CertifyX509_In*)MemoryGetInBuffer(sizeof(CertifyX509_In));
    CertifyX509_Out* out =
        (CertifyX509_Out*)MemoryGetOutBuffer(sizeof(CertifyX509_Out));
    in->objectHandle = handles[0];
    in->signHandle   = handles[1];
    result = TPM2B_DATA_Unmarshal(&in->reserved, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CertifyX509_reserved);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_CertifyX509_inScheme);
    result = TPM2B_MAX_BUFFER_Unmarshal(
        &in->partialCertificate, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CertifyX509_partialCertificate);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_CertifyX509(in, out);
    rSize  = sizeof(CertifyX509_Out);
    *respParmSize +=
        TPM2B_MAX_BUFFER_Marshal(&out->addedToCertificate, responseBuffer, &rSize);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->tbsDigest, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
#endif  // CC_CertifyX509
#if CC_Commit
case TPM_CC_Commit:
{
    Commit_In*  in  = (Commit_In*)MemoryGetInBuffer(sizeof(Commit_In));
    Commit_Out* out = (Commit_Out*)MemoryGetOutBuffer(sizeof(Commit_Out));
    in->signHandle  = handles[0];
    result = TPM2B_ECC_POINT_Unmarshal(&in->P1, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Commit_P1);
    result = TPM2B_SENSITIVE_DATA_Unmarshal(&in->s2, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Commit_s2);
    result = TPM2B_ECC_PARAMETER_Unmarshal(&in->y2, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Commit_y2);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Commit(in, out);
    rSize  = sizeof(Commit_Out);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->K, responseBuffer, &rSize);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->L, responseBuffer, &rSize);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->E, responseBuffer, &rSize);
    *respParmSize += UINT16_Marshal(&out->counter, responseBuffer, &rSize);
    break;
}
#endif  // CC_Commit
#if CC_EC_Ephemeral
case TPM_CC_EC_Ephemeral:
{
    EC_Ephemeral_In* in =
        (EC_Ephemeral_In*)MemoryGetInBuffer(sizeof(EC_Ephemeral_In));
```

```
    EC_Ephemeral_Out* out =
        (EC_Ephemeral_Out*)MemoryGetOutBuffer(sizeof(EC_Ephemeral_Out));
    result =
        TPMI_ECC_CURVE_Unmarshal(&in->curveID, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_EC_Ephemeral_curveID);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_EC_Ephemeral(in, out);
    rSize  = sizeof(EC_Ephemeral_Out);
    *respParmSize += TPM2B_ECC_POINT_Marshal(&out->Q, responseBuffer, &rSize);
    *respParmSize += UINT16_Marshal(&out->counter, responseBuffer, &rSize);
    break;
}
#endif  // CC_EC_Ephemeral
#if CC_VerifySignature
case TPM_CC_VerifySignature:
{
    VerifySignature_In* in =
        (VerifySignature_In*)MemoryGetInBuffer(sizeof(VerifySignature_In));
    VerifySignature_Out* out =
        (VerifySignature_Out*)MemoryGetOutBuffer(sizeof(VerifySignature_Out));
    in->keyHandle = handles[0];
    result        = TPM2B_DIGEST_Unmarshal(&in->digest, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_VerifySignature_digest);
    result =
        TPMT_SIGNATURE_Unmarshal(&in->signature, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_VerifySignature_signature);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_VerifySignature(in, out);
    rSize  = sizeof(VerifySignature_Out);
    *respParmSize +=
        TPMT_TK_VERIFIED_Marshal(&out->validation, responseBuffer, &rSize);
    break;
}
#endif  // CC_VerifySignature
#if CC_Sign
case TPM_CC_Sign:
{
    Sign_In*  in  = (Sign_In*)MemoryGetInBuffer(sizeof(Sign_In));
    Sign_Out* out = (Sign_Out*)MemoryGetOutBuffer(sizeof(Sign_Out));
    in->keyHandle = handles[0];
    result        = TPM2B_DIGEST_Unmarshal(&in->digest, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Sign_digest);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_Sign_inScheme);
    result =
        TPMT_TK_HASHCHECK_Unmarshal(&in->validation, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Sign_validation);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Sign(in, out);
    rSize  = sizeof(Sign_Out);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
```

```
#endif  // CC_Sign
#if CC_SetCommandCodeAuditStatus
case TPM_CC_SetCommandCodeAuditStatus:
{
    SetCommandCodeAuditStatus_In* in =
        (SetCommandCodeAuditStatus_In*)MemoryGetInBuffer(
            sizeof(SetCommandCodeAuditStatus_In));
    in->auth = handles[0];
    result =
        TPMI_ALG_HASH_Unmarshal(&in->auditAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_SetCommandCodeAuditStatus_auditAlg);
    result = TPML_CC_Unmarshal(&in->setList, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_SetCommandCodeAuditStatus_setList);
    result = TPML_CC_Unmarshal(&in->clearList, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_SetCommandCodeAuditStatus_clearList);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_SetCommandCodeAuditStatus(in);
    break;
}
#endif  // CC_SetCommandCodeAuditStatus
#if CC_PCR_Extend
case TPM_CC_PCR_Extend:
{
    PCR_Extend_In* in = (PCR_Extend_In*)MemoryGetInBuffer(sizeof(PCR_Extend_In));
    in->pcrHandle     = handles[0];
    result = TPML_DIGEST_VALUES_Unmarshal(&in->digests, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PCR_Extend_digests);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PCR_Extend(in);
    break;
}
#endif  // CC_PCR_Extend
#if CC_PCR_Event
case TPM_CC_PCR_Event:
{
    PCR_Event_In*  in  = (PCR_Event_In*)MemoryGetInBuffer(sizeof(PCR_Event_In));
    PCR_Event_Out* out = (PCR_Event_Out*)MemoryGetOutBuffer(sizeof(PCR_Event_Out));
    in->pcrHandle      = handles[0];
    result = TPM2B_EVENT_Unmarshal(&in->eventData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PCR_Event_eventData);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PCR_Event(in, out);
    rSize  = sizeof(PCR_Event_Out);
    *respParmSize +=
        TPML_DIGEST_VALUES_Marshal(&out->digests, responseBuffer, &rSize);
    break;
}
#endif  // CC_PCR_Event
#if CC_PCR_Read
case TPM_CC_PCR_Read:
{
    PCR_Read_In*  in  = (PCR_Read_In*)MemoryGetInBuffer(sizeof(PCR_Read_In));
    PCR_Read_Out* out = (PCR_Read_Out*)MemoryGetOutBuffer(sizeof(PCR_Read_Out));
    result            = TPML_PCR_SELECTION_Unmarshal(
```

```
        &in->pcrSelectionIn, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PCR_Read_pcrSelectionIn);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PCR_Read(in, out);
    rSize  = sizeof(PCR_Read_Out);
    *respParmSize += UINT32_Marshal(&out->pcrUpdateCounter, responseBuffer, &rSize);
    *respParmSize +=
        TPML_PCR_SELECTION_Marshal(&out->pcrSelectionOut, responseBuffer, &rSize);
    *respParmSize += TPML_DIGEST_Marshal(&out->pcrValues, responseBuffer, &rSize);
    break;
}
#endif  // CC_PCR_Read
#if CC_PCR_Allocate
case TPM_CC_PCR_Allocate:
{
    PCR_Allocate_In* in =
        (PCR_Allocate_In*)MemoryGetInBuffer(sizeof(PCR_Allocate_In));
    PCR_Allocate_Out* out =
        (PCR_Allocate_Out*)MemoryGetOutBuffer(sizeof(PCR_Allocate_Out));
    in->authHandle = handles[0];
    result         = TPML_PCR_SELECTION_Unmarshal(
        &in->pcrAllocation, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PCR_Allocate_pcrAllocation);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PCR_Allocate(in, out);
    rSize  = sizeof(PCR_Allocate_Out);
    *respParmSize +=
        TPMI_YES_NO_Marshal(&out->allocationSuccess, responseBuffer, &rSize);
    *respParmSize += UINT32_Marshal(&out->maxPCR, responseBuffer, &rSize);
    *respParmSize += UINT32_Marshal(&out->sizeNeeded, responseBuffer, &rSize);
    *respParmSize += UINT32_Marshal(&out->sizeAvailable, responseBuffer, &rSize);
    break;
}
#endif  // CC_PCR_Allocate
#if CC_PCR_SetAuthPolicy
case TPM_CC_PCR_SetAuthPolicy:
{
    PCR_SetAuthPolicy_In* in =
        (PCR_SetAuthPolicy_In*)MemoryGetInBuffer(sizeof(PCR_SetAuthPolicy_In));
    in->authHandle = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->authPolicy, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PCR_SetAuthPolicy_authPolicy);
    result =
        TPMI_ALG_HASH_Unmarshal(&in->hashAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_PCR_SetAuthPolicy_hashAlg);
    result = TPMI_DH_PCR_Unmarshal(&in->pcrNum, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_PCR_SetAuthPolicy_pcrNum);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PCR_SetAuthPolicy(in);
    break;
}
#endif  // CC_PCR_SetAuthPolicy
#if CC_PCR_SetAuthValue
case TPM_CC_PCR_SetAuthValue:
```

```
{
    PCR_SetAuthValue_In* in =
        (PCR_SetAuthValue_In*)MemoryGetInBuffer(sizeof(PCR_SetAuthValue_In));
    in->pcrHandle = handles[0];
    result        = TPM2B_DIGEST_Unmarshal(&in->auth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PCR_SetAuthValue_auth);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PCR_SetAuthValue(in);
    break;
}
#endif  // CC_PCR_SetAuthValue
#if CC_PCR_Reset
case TPM_CC_PCR_Reset:
{
    PCR_Reset_In* in = (PCR_Reset_In*)MemoryGetInBuffer(sizeof(PCR_Reset_In));
    in->pcrHandle    = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PCR_Reset(in);
    break;
}
#endif  // CC_PCR_Reset
#if CC_PolicySigned
case TPM_CC_PolicySigned:
{
    PolicySigned_In* in =
        (PolicySigned_In*)MemoryGetInBuffer(sizeof(PolicySigned_In));
    PolicySigned_Out* out =
        (PolicySigned_Out*)MemoryGetOutBuffer(sizeof(PolicySigned_Out));
    in->authObject    = handles[0];
    in->policySession = handles[1];
    result = TPM2B_NONCE_Unmarshal(&in->nonceTPM, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySigned_nonceTPM);
    result = TPM2B_DIGEST_Unmarshal(&in->cpHashA, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySigned_cpHashA);
    result = TPM2B_NONCE_Unmarshal(&in->policyRef, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySigned_policyRef);
    result = INT32_Unmarshal(&in->expiration, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySigned_expiration);
    result = TPMT_SIGNATURE_Unmarshal(&in->auth, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_PolicySigned_auth);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicySigned(in, out);
    rSize  = sizeof(PolicySigned_Out);
    *respParmSize += TPM2B_TIMEOUT_Marshal(&out->timeout, responseBuffer, &rSize);
    *respParmSize += TPMT_TK_AUTH_Marshal(&out->policyTicket, responseBuffer, &rSize);
    break;
}
#endif  // CC_PolicySigned
#if CC_PolicySecret
case TPM_CC_PolicySecret:
{
    PolicySecret_In* in =
        (PolicySecret_In*)MemoryGetInBuffer(sizeof(PolicySecret_In));
    PolicySecret_Out* out =
```

```
        (PolicySecret_Out*)MemoryGetOutBuffer(sizeof(PolicySecret_Out));
    in->authHandle    = handles[0];
    in->policySession = handles[1];
    result = TPM2B_NONCE_Unmarshal(&in->nonceTPM, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySecret_nonceTPM);
    result = TPM2B_DIGEST_Unmarshal(&in->cpHashA, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySecret_cpHashA);
    result = TPM2B_NONCE_Unmarshal(&in->policyRef, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySecret_policyRef);
    result = INT32_Unmarshal(&in->expiration, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicySecret_expiration);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicySecret(in, out);
    rSize  = sizeof(PolicySecret_Out);
    *respParmSize += TPM2B_TIMEOUT_Marshal(&out->timeout, responseBuffer, &rSize);
    *respParmSize += TPMT_TK_AUTH_Marshal(&out->policyTicket, responseBuffer, &rSize);
    break;
}
#endif  // CC_PolicySecret
#if CC_PolicyTicket
case TPM_CC_PolicyTicket:
{
    PolicyTicket_In* in =
        (PolicyTicket_In*)MemoryGetInBuffer(sizeof(PolicyTicket_In));
    in->policySession = handles[0];
    result = TPM2B_TIMEOUT_Unmarshal(&in->timeout, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyTicket_timeout);
    result = TPM2B_DIGEST_Unmarshal(&in->cpHashA, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyTicket_cpHashA);
    result = TPM2B_NONCE_Unmarshal(&in->policyRef, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyTicket_policyRef);
    result = TPM2B_NAME_Unmarshal(&in->authName, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyTicket_authName);
    result = TPMT_TK_AUTH_Unmarshal(&in->ticket, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyTicket_ticket);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyTicket(in);
    break;
}
#endif  // CC_PolicyTicket
#if CC_PolicyOR
case TPM_CC_PolicyOR:
{
    PolicyOR_In* in   = (PolicyOR_In*)MemoryGetInBuffer(sizeof(PolicyOR_In));
    in->policySession = handles[0];
    result = TPML_DIGEST_Unmarshal(&in->pHashList, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyOR_pHashList);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyOR(in);
    break;
}
#endif  // CC_PolicyOR
#if CC_PolicyPCR
case TPM_CC_PolicyPCR:
```

```
{
    PolicyPCR_In* in  = (PolicyPCR_In*)MemoryGetInBuffer(sizeof(PolicyPCR_In));
    in->policySession = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->pcrDigest, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyPCR_pcrDigest);
    result = TPML_PCR_SELECTION_Unmarshal(&in->pcrs, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyPCR_pcrs);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyPCR(in);
    break;
}
#endif  // CC_PolicyPCR
#if CC_PolicyLocality
case TPM_CC_PolicyLocality:
{
    PolicyLocality_In* in =
        (PolicyLocality_In*)MemoryGetInBuffer(sizeof(PolicyLocality_In));
    in->policySession = handles[0];
    result = TPMA_LOCALITY_Unmarshal(&in->locality, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyLocality_locality);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyLocality(in);
    break;
}
#endif  // CC_PolicyLocality
#if CC_PolicyNV
case TPM_CC_PolicyNV:
{
    PolicyNV_In* in   = (PolicyNV_In*)MemoryGetInBuffer(sizeof(PolicyNV_In));
    in->authHandle    = handles[0];
    in->nvIndex       = handles[1];
    in->policySession = handles[2];
    result = TPM2B_OPERAND_Unmarshal(&in->operandB, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyNV_operandB);
    result = UINT16_Unmarshal(&in->offset, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyNV_offset);
    result = TPM_EO_Unmarshal(&in->operation, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyNV_operation);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyNV(in);
    break;
}
#endif  // CC_PolicyNV
#if CC_PolicyCounterTimer
case TPM_CC_PolicyCounterTimer:
{
    PolicyCounterTimer_In* in =
        (PolicyCounterTimer_In*)MemoryGetInBuffer(sizeof(PolicyCounterTimer_In));
    in->policySession = handles[0];
    result = TPM2B_OPERAND_Unmarshal(&in->operandB, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCounterTimer_operandB);
    result = UINT16_Unmarshal(&in->offset, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCounterTimer_offset);
    result = TPM_EO_Unmarshal(&in->operation, paramBuffer, paramBufferSize);
```

```
    EXIT_IF_ERROR_PLUS(RC_PolicyCounterTimer_operation);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyCounterTimer(in);
    break;
}
#endif  // CC_PolicyCounterTimer
#if CC_PolicyCommandCode
case TPM_CC_PolicyCommandCode:
{
    PolicyCommandCode_In* in =
        (PolicyCommandCode_In*)MemoryGetInBuffer(sizeof(PolicyCommandCode_In));
    in->policySession = handles[0];
    result            = TPM_CC_Unmarshal(&in->code, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCommandCode_code);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyCommandCode(in);
    break;
}
#endif  // CC_PolicyCommandCode
#if CC_PolicyPhysicalPresence
case TPM_CC_PolicyPhysicalPresence:
{
    PolicyPhysicalPresence_In* in = (PolicyPhysicalPresence_In*)MemoryGetInBuffer(
        sizeof(PolicyPhysicalPresence_In));
    in->policySession = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyPhysicalPresence(in);
    break;
}
#endif  // CC_PolicyPhysicalPresence
#if CC_PolicyCpHash
case TPM_CC_PolicyCpHash:
{
    PolicyCpHash_In* in =
        (PolicyCpHash_In*)MemoryGetInBuffer(sizeof(PolicyCpHash_In));
    in->policySession = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->cpHashA, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCpHash_cpHashA);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyCpHash(in);
    break;
}
#endif  // CC_PolicyCpHash
#if CC_PolicyNameHash
case TPM_CC_PolicyNameHash:
{
    PolicyNameHash_In* in =
        (PolicyNameHash_In*)MemoryGetInBuffer(sizeof(PolicyNameHash_In));
    in->policySession = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->nameHash, paramBuffer, paramBufferSize);
```

```
    EXIT_IF_ERROR_PLUS(RC_PolicyNameHash_nameHash);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyNameHash(in);
    break;
}
#endif  // CC_PolicyNameHash
#if CC_PolicyDuplicationSelect
case TPM_CC_PolicyDuplicationSelect:
{
    PolicyDuplicationSelect_In* in = (PolicyDuplicationSelect_In*)MemoryGetInBuffer(
        sizeof(PolicyDuplicationSelect_In));
    in->policySession = handles[0];
    result = TPM2B_NAME_Unmarshal(&in->objectName, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyDuplicationSelect_objectName);
    result = TPM2B_NAME_Unmarshal(&in->newParentName, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyDuplicationSelect_newParentName);
    result = TPMI_YES_NO_Unmarshal(&in->includeObject, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyDuplicationSelect_includeObject);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyDuplicationSelect(in);
    break;
}
#endif  // CC_PolicyDuplicationSelect
#if CC_PolicyAuthorize
case TPM_CC_PolicyAuthorize:
{
    PolicyAuthorize_In* in =
        (PolicyAuthorize_In*)MemoryGetInBuffer(sizeof(PolicyAuthorize_In));
    in->policySession = handles[0];
    result =
        TPM2B_DIGEST_Unmarshal(&in->approvedPolicy, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyAuthorize_approvedPolicy);
    result = TPM2B_NONCE_Unmarshal(&in->policyRef, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyAuthorize_policyRef);
    result = TPM2B_NAME_Unmarshal(&in->keySign, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyAuthorize_keySign);
    result =
        TPMT_TK_VERIFIED_Unmarshal(&in->checkTicket, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyAuthorize_checkTicket);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyAuthorize(in);
    break;
}
#endif  // CC_PolicyAuthorize
#if CC_PolicyAuthValue
case TPM_CC_PolicyAuthValue:
{
    PolicyAuthValue_In* in =
        (PolicyAuthValue_In*)MemoryGetInBuffer(sizeof(PolicyAuthValue_In));
    in->policySession = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
```

```
    }
    result = TPM2_PolicyAuthValue(in);
    break;
}
#endif  // CC_PolicyAuthValue
#if CC_PolicyPassword
case TPM_CC_PolicyPassword:
{
    PolicyPassword_In* in =
        (PolicyPassword_In*)MemoryGetInBuffer(sizeof(PolicyPassword_In));
    in->policySession = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyPassword(in);
    break;
}
#endif  // CC_PolicyPassword
#if CC_PolicyGetDigest
case TPM_CC_PolicyGetDigest:
{
    PolicyGetDigest_In* in =
        (PolicyGetDigest_In*)MemoryGetInBuffer(sizeof(PolicyGetDigest_In));
    PolicyGetDigest_Out* out =
        (PolicyGetDigest_Out*)MemoryGetOutBuffer(sizeof(PolicyGetDigest_Out));
    in->policySession = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyGetDigest(in, out);
    rSize  = sizeof(PolicyGetDigest_Out);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->policyDigest, responseBuffer, &rSize);
    break;
}
#endif  // CC_PolicyGetDigest
#if CC_PolicyNvWritten
case TPM_CC_PolicyNvWritten:
{
    PolicyNvWritten_In* in =
        (PolicyNvWritten_In*)MemoryGetInBuffer(sizeof(PolicyNvWritten_In));
    in->policySession = handles[0];
    result = TPMI_YES_NO_Unmarshal(&in->writtenSet, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyNvWritten_writtenSet);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyNvWritten(in);
    break;
}
#endif  // CC_PolicyNvWritten
#if CC_PolicyTemplate
case TPM_CC_PolicyTemplate:
{
    PolicyTemplate_In* in =
        (PolicyTemplate_In*)MemoryGetInBuffer(sizeof(PolicyTemplate_In));
    in->policySession = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->templateHash, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyTemplate_templateHash);
    if(*paramBufferSize != 0)
    {
```

```
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyTemplate(in);
    break;
}
#endif  // CC_PolicyTemplate
#if CC_PolicyAuthorizeNV
case TPM_CC_PolicyAuthorizeNV:
{
    PolicyAuthorizeNV_In* in =
        (PolicyAuthorizeNV_In*)MemoryGetInBuffer(sizeof(PolicyAuthorizeNV_In));
    in->authHandle    = handles[0];
    in->nvIndex       = handles[1];
    in->policySession = handles[2];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyAuthorizeNV(in);
    break;
}
#endif  // CC_PolicyAuthorizeNV
#if CC_PolicyCapability
case TPM_CC_PolicyCapability:
{
    PolicyCapability_In* in =
        (PolicyCapability_In*)MemoryGetInBuffer(sizeof(PolicyCapability_In));
    in->policySession = handles[0];
    result = TPM2B_OPERAND_Unmarshal(&in->operandB, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCapability_operandB);
    result = UINT16_Unmarshal(&in->offset, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCapability_offset);
    result = TPM_EO_Unmarshal(&in->operation, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCapability_operation);
    result = TPM_CAP_Unmarshal(&in->capability, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCapability_capability);
    result = UINT32_Unmarshal(&in->property, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyCapability_property);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyCapability(in);
    break;
}
#endif  // CC_PolicyCapability
#if CC_PolicyParameters
case TPM_CC_PolicyParameters:
{
    PolicyParameters_In* in =
        (PolicyParameters_In*)MemoryGetInBuffer(sizeof(PolicyParameters_In));
    in->policySession = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->pHash, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PolicyParameters_pHash);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PolicyParameters(in);
    break;
}
#endif  // CC_PolicyParameters
```

```
#if CC_CreatePrimary
case TPM_CC_CreatePrimary:
{
    CreatePrimary_In* in =
        (CreatePrimary_In*)MemoryGetInBuffer(sizeof(CreatePrimary_In));
    CreatePrimary_Out* out =
        (CreatePrimary_Out*)MemoryGetOutBuffer(sizeof(CreatePrimary_Out));
    in->primaryHandle = handles[0];
    result            = TPM2B_SENSITIVE_CREATE_Unmarshal(
        &in->inSensitive, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CreatePrimary_inSensitive);
    result =
        TPM2B_PUBLIC_Unmarshal(&in->inPublic, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_CreatePrimary_inPublic);
    result = TPM2B_DATA_Unmarshal(&in->outsideInfo, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CreatePrimary_outsideInfo);
    result =
        TPML_PCR_SELECTION_Unmarshal(&in->creationPCR, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_CreatePrimary_creationPCR);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_CreatePrimary(in, out);
    rSize  = sizeof(CreatePrimary_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->objectHandle;
    *respParmSize += TPM2B_PUBLIC_Marshal(&out->outPublic, responseBuffer, &rSize);
    *respParmSize +=
        TPM2B_CREATION_DATA_Marshal(&out->creationData, responseBuffer, &rSize);
    *respParmSize += TPM2B_DIGEST_Marshal(&out->creationHash, responseBuffer, &rSize);
    *respParmSize +=
        TPMT_TK_CREATION_Marshal(&out->creationTicket, responseBuffer, &rSize);
    *respParmSize += TPM2B_NAME_Marshal(&out->name, responseBuffer, &rSize);
    break;
}
#endif  // CC_CreatePrimary
#if CC_HierarchyControl
case TPM_CC_HierarchyControl:
{
    HierarchyControl_In* in =
        (HierarchyControl_In*)MemoryGetInBuffer(sizeof(HierarchyControl_In));
    in->authHandle = handles[0];
    result =
        TPMI_RH_ENABLES_Unmarshal(&in->enable, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_HierarchyControl_enable);
    result = TPMI_YES_NO_Unmarshal(&in->state, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_HierarchyControl_state);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_HierarchyControl(in);
    break;
}
#endif  // CC_HierarchyControl
#if CC_SetPrimaryPolicy
case TPM_CC_SetPrimaryPolicy:
{
    SetPrimaryPolicy_In* in =
        (SetPrimaryPolicy_In*)MemoryGetInBuffer(sizeof(SetPrimaryPolicy_In));
    in->authHandle = handles[0];
    result = TPM2B_DIGEST_Unmarshal(&in->authPolicy, paramBuffer, paramBufferSize);
```

```
    EXIT_IF_ERROR_PLUS(RC_SetPrimaryPolicy_authPolicy);
    result =
        TPMI_ALG_HASH_Unmarshal(&in->hashAlg, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_SetPrimaryPolicy_hashAlg);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_SetPrimaryPolicy(in);
    break;
}
#endif  // CC_SetPrimaryPolicy
#if CC_ChangePPS
case TPM_CC_ChangePPS:
{
    ChangePPS_In* in = (ChangePPS_In*)MemoryGetInBuffer(sizeof(ChangePPS_In));
    in->authHandle   = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ChangePPS(in);
    break;
}
#endif  // CC_ChangePPS
#if CC_ChangeEPS
case TPM_CC_ChangeEPS:
{
    ChangeEPS_In* in = (ChangeEPS_In*)MemoryGetInBuffer(sizeof(ChangeEPS_In));
    in->authHandle   = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ChangeEPS(in);
    break;
}
#endif  // CC_ChangeEPS
#if CC_Clear
case TPM_CC_Clear:
{
    Clear_In* in   = (Clear_In*)MemoryGetInBuffer(sizeof(Clear_In));
    in->authHandle = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Clear(in);
    break;
}
#endif  // CC_Clear
#if CC_ClearControl
case TPM_CC_ClearControl:
{
    ClearControl_In* in =
        (ClearControl_In*)MemoryGetInBuffer(sizeof(ClearControl_In));
    in->auth = handles[0];
    result   = TPMI_YES_NO_Unmarshal(&in->disable, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ClearControl_disable);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
```

```
        goto Exit;
    }
    result = TPM2_ClearControl(in);
    break;
}
#endif  // CC_ClearControl
#if CC_HierarchyChangeAuth
case TPM_CC_HierarchyChangeAuth:
{
    HierarchyChangeAuth_In* in =
        (HierarchyChangeAuth_In*)MemoryGetInBuffer(sizeof(HierarchyChangeAuth_In));
    in->authHandle = handles[0];
    result         = TPM2B_AUTH_Unmarshal(&in->newAuth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_HierarchyChangeAuth_newAuth);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_HierarchyChangeAuth(in);
    break;
}
#endif  // CC_HierarchyChangeAuth
#if CC_DictionaryAttackLockReset
case TPM_CC_DictionaryAttackLockReset:
{
    DictionaryAttackLockReset_In* in =
        (DictionaryAttackLockReset_In*)MemoryGetInBuffer(
            sizeof(DictionaryAttackLockReset_In));
    in->lockHandle = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_DictionaryAttackLockReset(in);
    break;
}
#endif  // CC_DictionaryAttackLockReset
#if CC_DictionaryAttackParameters
case TPM_CC_DictionaryAttackParameters:
{
    DictionaryAttackParameters_In* in =
        (DictionaryAttackParameters_In*)MemoryGetInBuffer(
            sizeof(DictionaryAttackParameters_In));
    in->lockHandle = handles[0];
    result         = UINT32_Unmarshal(&in->newMaxTries, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_DictionaryAttackParameters_newMaxTries);
    result = UINT32_Unmarshal(&in->newRecoveryTime, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_DictionaryAttackParameters_newRecoveryTime);
    result = UINT32_Unmarshal(&in->lockoutRecovery, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_DictionaryAttackParameters_lockoutRecovery);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_DictionaryAttackParameters(in);
    break;
}
#endif  // CC_DictionaryAttackParameters
#if CC_PP_Commands
case TPM_CC_PP_Commands:
{
    PP_Commands_In* in = (PP_Commands_In*)MemoryGetInBuffer(sizeof(PP_Commands_In));
    in->auth           = handles[0];
```

```
    result = TPML_CC_Unmarshal(&in->setList, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PP_Commands_setList);
    result = TPML_CC_Unmarshal(&in->clearList, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_PP_Commands_clearList);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_PP_Commands(in);
    break;
}
#endif  // CC_PP_Commands
#if CC_SetAlgorithmSet
case TPM_CC_SetAlgorithmSet:
{
    SetAlgorithmSet_In* in =
        (SetAlgorithmSet_In*)MemoryGetInBuffer(sizeof(SetAlgorithmSet_In));
    in->authHandle = handles[0];
    result = UINT32_Unmarshal(&in->algorithmSet, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_SetAlgorithmSet_algorithmSet);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_SetAlgorithmSet(in);
    break;
}
#endif  // CC_SetAlgorithmSet
#if CC_FieldUpgradeStart
case TPM_CC_FieldUpgradeStart:
{
    FieldUpgradeStart_In* in =
        (FieldUpgradeStart_In*)MemoryGetInBuffer(sizeof(FieldUpgradeStart_In));
    in->authorization = handles[0];
    in->keyHandle     = handles[1];
    result = TPM2B_DIGEST_Unmarshal(&in->fuDigest, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_FieldUpgradeStart_fuDigest);
    result = TPMT_SIGNATURE_Unmarshal(
        &in->manifestSignature, paramBuffer, paramBufferSize, FALSE);
    EXIT_IF_ERROR_PLUS(RC_FieldUpgradeStart_manifestSignature);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_FieldUpgradeStart(in);
    break;
}
#endif  // CC_FieldUpgradeStart
#if CC_FieldUpgradeData
case TPM_CC_FieldUpgradeData:
{
    FieldUpgradeData_In* in =
        (FieldUpgradeData_In*)MemoryGetInBuffer(sizeof(FieldUpgradeData_In));
    FieldUpgradeData_Out* out =
        (FieldUpgradeData_Out*)MemoryGetOutBuffer(sizeof(FieldUpgradeData_Out));
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->fuData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_FieldUpgradeData_fuData);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_FieldUpgradeData(in, out);
```

```
    rSize  = sizeof(FieldUpgradeData_Out);
    *respParmSize += TPMT_HA_Marshal(&out->nextDigest, responseBuffer, &rSize);
    *respParmSize += TPMT_HA_Marshal(&out->firstDigest, responseBuffer, &rSize);
    break;
}
#endif  // CC_FieldUpgradeData
#if CC_FirmwareRead
case TPM_CC_FirmwareRead:
{
    FirmwareRead_In* in =
        (FirmwareRead_In*)MemoryGetInBuffer(sizeof(FirmwareRead_In));
    FirmwareRead_Out* out =
        (FirmwareRead_Out*)MemoryGetOutBuffer(sizeof(FirmwareRead_Out));
    result = UINT32_Unmarshal(&in->sequenceNumber, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_FirmwareRead_sequenceNumber);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_FirmwareRead(in, out);
    rSize  = sizeof(FirmwareRead_Out);
    *respParmSize += TPM2B_MAX_BUFFER_Marshal(&out->fuData, responseBuffer, &rSize);
    break;
}
#endif  // CC_FirmwareRead
#if CC_ContextSave
case TPM_CC_ContextSave:
{
    ContextSave_In*  in = (ContextSave_In*)MemoryGetInBuffer(sizeof(ContextSave_In));
    ContextSave_Out* out =
        (ContextSave_Out*)MemoryGetOutBuffer(sizeof(ContextSave_Out));
    in->saveHandle = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ContextSave(in, out);
    rSize  = sizeof(ContextSave_Out);
    *respParmSize += TPMS_CONTEXT_Marshal(&out->context, responseBuffer, &rSize);
    break;
}
#endif  // CC_ContextSave
#if CC_ContextLoad
case TPM_CC_ContextLoad:
{
    ContextLoad_In*  in = (ContextLoad_In*)MemoryGetInBuffer(sizeof(ContextLoad_In));
    ContextLoad_Out* out =
        (ContextLoad_Out*)MemoryGetOutBuffer(sizeof(ContextLoad_Out));
    result = TPMS_CONTEXT_Unmarshal(&in->context, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ContextLoad_context);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ContextLoad(in, out);
    rSize  = sizeof(ContextLoad_Out);
    if(TPM_RC_SUCCESS != result)
        goto Exit;
    command->handles[command->handleNum++] = out->loadedHandle;
    break;
}
#endif  // CC_ContextLoad
#if CC_FlushContext
```

```
case TPM_CC_FlushContext:
{
    FlushContext_In* in =
        (FlushContext_In*)MemoryGetInBuffer(sizeof(FlushContext_In));
    result =
        TPMI_DH_CONTEXT_Unmarshal(&in->flushHandle, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_FlushContext_flushHandle);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_FlushContext(in);
    break;
}
#endif  // CC_FlushContext
#if CC_EvictControl
case TPM_CC_EvictControl:
{
    EvictControl_In* in =
        (EvictControl_In*)MemoryGetInBuffer(sizeof(EvictControl_In));
    in->auth         = handles[0];
    in->objectHandle = handles[1];
    result           = TPMI_DH_PERSISTENT_Unmarshal(
        &in->persistentHandle, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_EvictControl_persistentHandle);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_EvictControl(in);
    break;
}
#endif  // CC_EvictControl
#if CC_ReadClock
case TPM_CC_ReadClock:
{
    ReadClock_Out* out = (ReadClock_Out*)MemoryGetOutBuffer(sizeof(ReadClock_Out));
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ReadClock(out);
    rSize  = sizeof(ReadClock_Out);
    *respParmSize +=
        TPMS_TIME_INFO_Marshal(&out->currentTime, responseBuffer, &rSize);
    break;
}
#endif  // CC_ReadClock
#if CC_ClockSet
case TPM_CC_ClockSet:
{
    ClockSet_In* in = (ClockSet_In*)MemoryGetInBuffer(sizeof(ClockSet_In));
    in->auth        = handles[0];
    result          = UINT64_Unmarshal(&in->newTime, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ClockSet_newTime);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ClockSet(in);
    break;
}
```

```
#endif  // CC_ClockSet
#if CC_ClockRateAdjust
case TPM_CC_ClockRateAdjust:
{
    ClockRateAdjust_In* in =
        (ClockRateAdjust_In*)MemoryGetInBuffer(sizeof(ClockRateAdjust_In));
    in->auth = handles[0];
    result =
        TPM_CLOCK_ADJUST_Unmarshal(&in->rateAdjust, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ClockRateAdjust_rateAdjust);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ClockRateAdjust(in);
    break;
}
#endif  // CC_ClockRateAdjust
#if CC_GetCapability
case TPM_CC_GetCapability:
{
    GetCapability_In* in =
        (GetCapability_In*)MemoryGetInBuffer(sizeof(GetCapability_In));
    GetCapability_Out* out =
        (GetCapability_Out*)MemoryGetOutBuffer(sizeof(GetCapability_Out));
    result = TPM_CAP_Unmarshal(&in->capability, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_GetCapability_capability);
    result = UINT32_Unmarshal(&in->property, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_GetCapability_property);
    result = UINT32_Unmarshal(&in->propertyCount, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_GetCapability_propertyCount);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_GetCapability(in, out);
    rSize  = sizeof(GetCapability_Out);
    *respParmSize += TPMI_YES_NO_Marshal(&out->moreData, responseBuffer, &rSize);
    *respParmSize +=
        TPMS_CAPABILITY_DATA_Marshal(&out->capabilityData, responseBuffer, &rSize);
    break;
}
#endif  // CC_GetCapability
#if CC_TestParms
case TPM_CC_TestParms:
{
    TestParms_In* in = (TestParms_In*)MemoryGetInBuffer(sizeof(TestParms_In));
    result =
        TPMT_PUBLIC_PARMS_Unmarshal(&in->parameters, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_TestParms_parameters);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_TestParms(in);
    break;
}
#endif  // CC_TestParms
#if CC_NV_DefineSpace
case TPM_CC_NV_DefineSpace:
{
    NV_DefineSpace_In* in =
        (NV_DefineSpace_In*)MemoryGetInBuffer(sizeof(NV_DefineSpace_In));
```

```
    in->authHandle = handles[0];
    result         = TPM2B_AUTH_Unmarshal(&in->auth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_DefineSpace_auth);
    result = TPM2B_NV_PUBLIC_Unmarshal(&in->publicInfo, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_DefineSpace_publicInfo);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_DefineSpace(in);
    break;
}
#endif  // CC_NV_DefineSpace
#if CC_NV_UndefineSpace
case TPM_CC_NV_UndefineSpace:
{
    NV_UndefineSpace_In* in =
        (NV_UndefineSpace_In*)MemoryGetInBuffer(sizeof(NV_UndefineSpace_In));
    in->authHandle = handles[0];
    in->nvIndex    = handles[1];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_UndefineSpace(in);
    break;
}
#endif  // CC_NV_UndefineSpace
#if CC_NV_UndefineSpaceSpecial
case TPM_CC_NV_UndefineSpaceSpecial:
{
    NV_UndefineSpaceSpecial_In* in = (NV_UndefineSpaceSpecial_In*)MemoryGetInBuffer(
        sizeof(NV_UndefineSpaceSpecial_In));
    in->nvIndex  = handles[0];
    in->platform = handles[1];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_UndefineSpaceSpecial(in);
    break;
}
#endif  // CC_NV_UndefineSpaceSpecial
#if CC_NV_ReadPublic
case TPM_CC_NV_ReadPublic:
{
    NV_ReadPublic_In* in =
        (NV_ReadPublic_In*)MemoryGetInBuffer(sizeof(NV_ReadPublic_In));
    NV_ReadPublic_Out* out =
        (NV_ReadPublic_Out*)MemoryGetOutBuffer(sizeof(NV_ReadPublic_Out));
    in->nvIndex = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_ReadPublic(in, out);
    rSize  = sizeof(NV_ReadPublic_Out);
    *respParmSize += TPM2B_NV_PUBLIC_Marshal(&out->nvPublic, responseBuffer, &rSize);
    *respParmSize += TPM2B_NAME_Marshal(&out->nvName, responseBuffer, &rSize);
    break;
}
#endif  // CC_NV_ReadPublic
```

```
#if CC_NV_Write
case TPM_CC_NV_Write:
{
    NV_Write_In* in = (NV_Write_In*)MemoryGetInBuffer(sizeof(NV_Write_In));
    in->authHandle  = handles[0];
    in->nvIndex     = handles[1];
    result = TPM2B_MAX_NV_BUFFER_Unmarshal(&in->data, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Write_data);
    result = UINT16_Unmarshal(&in->offset, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Write_offset);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_Write(in);
    break;
}
#endif  // CC_NV_Write
#if CC_NV_Increment
case TPM_CC_NV_Increment:
{
    NV_Increment_In* in =
        (NV_Increment_In*)MemoryGetInBuffer(sizeof(NV_Increment_In));
    in->authHandle = handles[0];
    in->nvIndex    = handles[1];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_Increment(in);
    break;
}
#endif  // CC_NV_Increment
#if CC_NV_Extend
case TPM_CC_NV_Extend:
{
    NV_Extend_In* in = (NV_Extend_In*)MemoryGetInBuffer(sizeof(NV_Extend_In));
    in->authHandle   = handles[0];
    in->nvIndex      = handles[1];
    result = TPM2B_MAX_NV_BUFFER_Unmarshal(&in->data, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Extend_data);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_Extend(in);
    break;
}
#endif  // CC_NV_Extend
#if CC_NV_SetBits
case TPM_CC_NV_SetBits:
{
    NV_SetBits_In* in = (NV_SetBits_In*)MemoryGetInBuffer(sizeof(NV_SetBits_In));
    in->authHandle    = handles[0];
    in->nvIndex       = handles[1];
    result            = UINT64_Unmarshal(&in->bits, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_SetBits_bits);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_SetBits(in);
```

```
    break;
}
#endif  // CC_NV_SetBits
#if CC_NV_WriteLock
case TPM_CC_NV_WriteLock:
{
    NV_WriteLock_In* in =
        (NV_WriteLock_In*)MemoryGetInBuffer(sizeof(NV_WriteLock_In));
    in->authHandle = handles[0];
    in->nvIndex    = handles[1];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_WriteLock(in);
    break;
}
#endif  // CC_NV_WriteLock
#if CC_NV_GlobalWriteLock
case TPM_CC_NV_GlobalWriteLock:
{
    NV_GlobalWriteLock_In* in =
        (NV_GlobalWriteLock_In*)MemoryGetInBuffer(sizeof(NV_GlobalWriteLock_In));
    in->authHandle = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_GlobalWriteLock(in);
    break;
}
#endif  // CC_NV_GlobalWriteLock
#if CC_NV_Read
case TPM_CC_NV_Read:
{
    NV_Read_In*  in  = (NV_Read_In*)MemoryGetInBuffer(sizeof(NV_Read_In));
    NV_Read_Out* out = (NV_Read_Out*)MemoryGetOutBuffer(sizeof(NV_Read_Out));
    in->authHandle   = handles[0];
    in->nvIndex      = handles[1];
    result           = UINT16_Unmarshal(&in->size, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Read_size);
    result = UINT16_Unmarshal(&in->offset, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Read_offset);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_Read(in, out);
    rSize  = sizeof(NV_Read_Out);
    *respParmSize += TPM2B_MAX_NV_BUFFER_Marshal(&out->data, responseBuffer, &rSize);
    break;
}
#endif  // CC_NV_Read
#if CC_NV_ReadLock
case TPM_CC_NV_ReadLock:
{
    NV_ReadLock_In* in = (NV_ReadLock_In*)MemoryGetInBuffer(sizeof(NV_ReadLock_In));
    in->authHandle     = handles[0];
    in->nvIndex        = handles[1];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
```

```
    }
    result = TPM2_NV_ReadLock(in);
    break;
}
#endif  // CC_NV_ReadLock
#if CC_NV_ChangeAuth
case TPM_CC_NV_ChangeAuth:
{
    NV_ChangeAuth_In* in =
        (NV_ChangeAuth_In*)MemoryGetInBuffer(sizeof(NV_ChangeAuth_In));
    in->nvIndex = handles[0];
    result      = TPM2B_AUTH_Unmarshal(&in->newAuth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_ChangeAuth_newAuth);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_ChangeAuth(in);
    break;
}
#endif  // CC_NV_ChangeAuth
#if CC_NV_Certify
case TPM_CC_NV_Certify:
{
    NV_Certify_In*  in  = (NV_Certify_In*)MemoryGetInBuffer(sizeof(NV_Certify_In));
    NV_Certify_Out* out = (NV_Certify_Out*)MemoryGetOutBuffer(sizeof(NV_Certify_Out));
    in->signHandle      = handles[0];
    in->authHandle      = handles[1];
    in->nvIndex         = handles[2];
    result = TPM2B_DATA_Unmarshal(&in->qualifyingData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Certify_qualifyingData);
    result =
        TPMT_SIG_SCHEME_Unmarshal(&in->inScheme, paramBuffer, paramBufferSize, TRUE);
    EXIT_IF_ERROR_PLUS(RC_NV_Certify_inScheme);
    result = UINT16_Unmarshal(&in->size, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Certify_size);
    result = UINT16_Unmarshal(&in->offset, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_Certify_offset);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_Certify(in, out);
    rSize  = sizeof(NV_Certify_Out);
    *respParmSize += TPM2B_ATTEST_Marshal(&out->certifyInfo, responseBuffer, &rSize);
    *respParmSize += TPMT_SIGNATURE_Marshal(&out->signature, responseBuffer, &rSize);
    break;
}
#endif  // CC_NV_Certify
#if CC_NV_DefineSpace2
case TPM_CC_NV_DefineSpace2:
{
    NV_DefineSpace2_In* in =
        (NV_DefineSpace2_In*)MemoryGetInBuffer(sizeof(NV_DefineSpace2_In));
    in->authHandle = handles[0];
    result         = TPM2B_AUTH_Unmarshal(&in->auth, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_DefineSpace2_auth);
    result =
        TPM2B_NV_PUBLIC_2_Unmarshal(&in->publicInfo, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_NV_DefineSpace2_publicInfo);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
```

```
    }
    result = TPM2_NV_DefineSpace2(in);
    break;
}
#endif  // CC_NV_DefineSpace2
#if CC_NV_ReadPublic2
case TPM_CC_NV_ReadPublic2:
{
    NV_ReadPublic2_In* in =
        (NV_ReadPublic2_In*)MemoryGetInBuffer(sizeof(NV_ReadPublic2_In));
    NV_ReadPublic2_Out* out =
        (NV_ReadPublic2_Out*)MemoryGetOutBuffer(sizeof(NV_ReadPublic2_Out));
    in->nvIndex = handles[0];
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_NV_ReadPublic2(in, out);
    rSize  = sizeof(NV_ReadPublic2_Out);
    *respParmSize +=
        TPM2B_NV_PUBLIC_2_Marshal(&out->nvPublic, responseBuffer, &rSize);
    *respParmSize += TPM2B_NAME_Marshal(&out->nvName, responseBuffer, &rSize);
    break;
}
#endif  // CC_NV_ReadPublic2
#if CC_SetCapability
case TPM_CC_SetCapability:
{
    SetCapability_In* in =
        (SetCapability_In*)MemoryGetInBuffer(sizeof(SetCapability_In));
    SetCapability_Out* out =
        (SetCapability_Out*)MemoryGetOutBuffer(sizeof(SetCapability_Out));
    in->authHandle = handles[0];
    result         = TPM2B_SET_CAPABILITY_DATA_Unmarshal(
        &in->setCapabilityData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_SetCapability_setCapabilityData);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_SetCapability(in);
    break;
}
#endif  // CC_SetCapability
#if CC_AC_GetCapability
case TPM_CC_AC_GetCapability:
{
    AC_GetCapability_In* in =
        (AC_GetCapability_In*)MemoryGetInBuffer(sizeof(AC_GetCapability_In));
    AC_GetCapability_Out* out =
        (AC_GetCapability_Out*)MemoryGetOutBuffer(sizeof(AC_GetCapability_Out));
    in->ac = handles[0];
    result = TPM_AT_Unmarshal(&in->capability, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_AC_GetCapability_capability);
    result = UINT32_Unmarshal(&in->count, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_AC_GetCapability_count);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_AC_GetCapability(in, out);
    rSize  = sizeof(AC_GetCapability_Out);
    *respParmSize += TPMI_YES_NO_Marshal(&out->moreData, responseBuffer, &rSize);
```

```
    *respParmSize +=
        TPML_AC_CAPABILITIES_Marshal(&out->capabilitiesData, responseBuffer, &rSize);
    break;
}
#endif  // CC_AC_GetCapability
#if CC_AC_Send
case TPM_CC_AC_Send:
{
    AC_Send_In*  in  = (AC_Send_In*)MemoryGetInBuffer(sizeof(AC_Send_In));
    AC_Send_Out* out = (AC_Send_Out*)MemoryGetOutBuffer(sizeof(AC_Send_Out));
    in->sendObject   = handles[0];
    in->authHandle   = handles[1];
    in->ac           = handles[2];
    result = TPM2B_MAX_BUFFER_Unmarshal(&in->acDataIn, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_AC_Send_acDataIn);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_AC_Send(in, out);
    rSize  = sizeof(AC_Send_Out);
    *respParmSize += TPMS_AC_OUTPUT_Marshal(&out->acDataOut, responseBuffer, &rSize);
    break;
}
#endif  // CC_AC_Send
#if CC_Policy_AC_SendSelect
case TPM_CC_Policy_AC_SendSelect:
{
    Policy_AC_SendSelect_In* in =
        (Policy_AC_SendSelect_In*)MemoryGetInBuffer(sizeof(Policy_AC_SendSelect_In));
    in->policySession = handles[0];
    result = TPM2B_NAME_Unmarshal(&in->objectName, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Policy_AC_SendSelect_objectName);
    result = TPM2B_NAME_Unmarshal(&in->authHandleName, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Policy_AC_SendSelect_authHandleName);
    result = TPM2B_NAME_Unmarshal(&in->acName, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Policy_AC_SendSelect_acName);
    result = TPMI_YES_NO_Unmarshal(&in->includeObject, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Policy_AC_SendSelect_includeObject);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Policy_AC_SendSelect(in);
    break;
}
#endif  // CC_Policy_AC_SendSelect
#if CC_ACT_SetTimeout
case TPM_CC_ACT_SetTimeout:
{
    ACT_SetTimeout_In* in =
        (ACT_SetTimeout_In*)MemoryGetInBuffer(sizeof(ACT_SetTimeout_In));
    in->actHandle = handles[0];
    result        = UINT32_Unmarshal(&in->startTimeout, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_ACT_SetTimeout_startTimeout);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_ACT_SetTimeout(in);
    break;
}
#endif  // CC_ACT_SetTimeout
```

```
#if CC_Vendor_TCG_Test
case TPM_CC_Vendor_TCG_Test:
{
    Vendor_TCG_Test_In* in =
        (Vendor_TCG_Test_In*)MemoryGetInBuffer(sizeof(Vendor_TCG_Test_In));
    Vendor_TCG_Test_Out* out =
        (Vendor_TCG_Test_Out*)MemoryGetOutBuffer(sizeof(Vendor_TCG_Test_Out));
    result = TPM2B_DATA_Unmarshal(&in->inputData, paramBuffer, paramBufferSize);
    EXIT_IF_ERROR_PLUS(RC_Vendor_TCG_Test_inputData);
    if(*paramBufferSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
    result = TPM2_Vendor_TCG_Test(in, out);
    rSize  = sizeof(Vendor_TCG_Test_Out);
    *respParmSize += TPM2B_DATA_Marshal(&out->outputData, responseBuffer, &rSize);
    break;
}
#endif  // CC_Vendor_TCG_Test
```

6.15 /tpm/include/private/Commands.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#ifndef _COMMANDS_H_
#define _COMMANDS_H_

#if CC_Startup
#  include "Startup_fp.h"
#endif
#if CC_Shutdown
#  include "Shutdown_fp.h"
#endif
#if CC_SelfTest
#  include "SelfTest_fp.h"
#endif
#if CC_IncrementalSelfTest
#  include "IncrementalSelfTest_fp.h"
#endif
#if CC_GetTestResult
#  include "GetTestResult_fp.h"
#endif
#if CC_StartAuthSession
#  include "StartAuthSession_fp.h"
#endif
#if CC_PolicyRestart
#  include "PolicyRestart_fp.h"
#endif
#if CC_Create
#  include "Create_fp.h"
#endif
#if CC_Load
#  include "Load_fp.h"
#endif
#if CC_LoadExternal
#  include "LoadExternal_fp.h"
#endif
#if CC_ReadPublic
#  include "ReadPublic_fp.h"
#endif
#if CC_ActivateCredential
#  include "ActivateCredential_fp.h"
#endif
#if CC_MakeCredential
```

```
#  include "MakeCredential_fp.h"
#endif
#if CC_Unseal
#  include "Unseal_fp.h"
#endif
#if CC_ObjectChangeAuth
#  include "ObjectChangeAuth_fp.h"
#endif
#if CC_CreateLoaded
#  include "CreateLoaded_fp.h"
#endif
#if CC_Duplicate
#  include "Duplicate_fp.h"
#endif
#if CC_Rewrap
#  include "Rewrap_fp.h"
#endif
#if CC_Import
#  include "Import_fp.h"
#endif
#if CC_RSA_Encrypt
#  include "RSA_Encrypt_fp.h"
#endif
#if CC_RSA_Decrypt
#  include "RSA_Decrypt_fp.h"
#endif
#if CC_ECDH_KeyGen
#  include "ECDH_KeyGen_fp.h"
#endif
#if CC_ECDH_ZGen
#  include "ECDH_ZGen_fp.h"
#endif
#if CC_ECC_Parameters
#  include "ECC_Parameters_fp.h"
#endif
#if CC_ZGen_2Phase
#  include "ZGen_2Phase_fp.h"
#endif
#if CC_ECC_Encrypt
#  include "ECC_Encrypt_fp.h"
#endif
#if CC_ECC_Decrypt
#  include "ECC_Decrypt_fp.h"
#endif
#if CC_EncryptDecrypt
#  include "EncryptDecrypt_fp.h"
#endif
#if CC_EncryptDecrypt2
#  include "EncryptDecrypt2_fp.h"
#endif
#if CC_Hash
#  include "Hash_fp.h"
#endif
#if CC_HMAC
#  include "HMAC_fp.h"
#endif
#if CC_MAC
#  include "MAC_fp.h"
#endif
#if CC_GetRandom
#  include "GetRandom_fp.h"
#endif
#if CC_StirRandom
#  include "StirRandom_fp.h"
#endif
#if CC_HMAC_Start
```

```
#  include "HMAC_Start_fp.h"
#endif
#if CC_MAC_Start
#  include "MAC_Start_fp.h"
#endif
#if CC_HashSequenceStart
#  include "HashSequenceStart_fp.h"
#endif
#if CC_SequenceUpdate
#  include "SequenceUpdate_fp.h"
#endif
#if CC_SequenceComplete
#  include "SequenceComplete_fp.h"
#endif
#if CC_EventSequenceComplete
#  include "EventSequenceComplete_fp.h"
#endif
#if CC_Certify
#  include "Certify_fp.h"
#endif
#if CC_CertifyCreation
#  include "CertifyCreation_fp.h"
#endif
#if CC_Quote
#  include "Quote_fp.h"
#endif
#if CC_GetSessionAuditDigest
#  include "GetSessionAuditDigest_fp.h"
#endif
#if CC_GetCommandAuditDigest
#  include "GetCommandAuditDigest_fp.h"
#endif
#if CC_GetTime
#  include "GetTime_fp.h"
#endif
#if CC_CertifyX509
#  include "CertifyX509_fp.h"
#endif
#if CC_Commit
#  include "Commit_fp.h"
#endif
#if CC_EC_Ephemeral
#  include "EC_Ephemeral_fp.h"
#endif
#if CC_VerifySignature
#  include "VerifySignature_fp.h"
#endif
#if CC_Sign
#  include "Sign_fp.h"
#endif
#if CC_SetCommandCodeAuditStatus
#  include "SetCommandCodeAuditStatus_fp.h"
#endif
#if CC_PCR_Extend
#  include "PCR_Extend_fp.h"
#endif
#if CC_PCR_Event
#  include "PCR_Event_fp.h"
#endif
#if CC_PCR_Read
#  include "PCR_Read_fp.h"
#endif
#if CC_PCR_Allocate
#  include "PCR_Allocate_fp.h"
#endif
#if CC_PCR_SetAuthPolicy
```

```
#  include "PCR_SetAuthPolicy_fp.h"
#endif
#if CC_PCR_SetAuthValue
#  include "PCR_SetAuthValue_fp.h"
#endif
#if CC_PCR_Reset
#  include "PCR_Reset_fp.h"
#endif
#if CC_PolicySigned
#  include "PolicySigned_fp.h"
#endif
#if CC_PolicySecret
#  include "PolicySecret_fp.h"
#endif
#if CC_PolicyTicket
#  include "PolicyTicket_fp.h"
#endif
#if CC_PolicyOR
#  include "PolicyOR_fp.h"
#endif
#if CC_PolicyPCR
#  include "PolicyPCR_fp.h"
#endif
#if CC_PolicyLocality
#  include "PolicyLocality_fp.h"
#endif
#if CC_PolicyNV
#  include "PolicyNV_fp.h"
#endif
#if CC_PolicyCounterTimer
#  include "PolicyCounterTimer_fp.h"
#endif
#if CC_PolicyCommandCode
#  include "PolicyCommandCode_fp.h"
#endif
#if CC_PolicyPhysicalPresence
#  include "PolicyPhysicalPresence_fp.h"
#endif
#if CC_PolicyCpHash
#  include "PolicyCpHash_fp.h"
#endif
#if CC_PolicyNameHash
#  include "PolicyNameHash_fp.h"
#endif
#if CC_PolicyDuplicationSelect
#  include "PolicyDuplicationSelect_fp.h"
#endif
#if CC_PolicyAuthorize
#  include "PolicyAuthorize_fp.h"
#endif
#if CC_PolicyAuthValue
#  include "PolicyAuthValue_fp.h"
#endif
#if CC_PolicyPassword
#  include "PolicyPassword_fp.h"
#endif
#if CC_PolicyGetDigest
#  include "PolicyGetDigest_fp.h"
#endif
#if CC_PolicyNvWritten
#  include "PolicyNvWritten_fp.h"
#endif
#if CC_PolicyTemplate
#  include "PolicyTemplate_fp.h"
#endif
#if CC_PolicyAuthorizeNV
```

```
#  include "PolicyAuthorizeNV_fp.h"
#endif
#if CC_PolicyCapability
#  include "PolicyCapability_fp.h"
#endif
#if CC_PolicyParameters
#  include "PolicyParameters_fp.h"
#endif
#if CC_CreatePrimary
#  include "CreatePrimary_fp.h"
#endif
#if CC_HierarchyControl
#  include "HierarchyControl_fp.h"
#endif
#if CC_SetPrimaryPolicy
#  include "SetPrimaryPolicy_fp.h"
#endif
#if CC_ChangePPS
#  include "ChangePPS_fp.h"
#endif
#if CC_ChangeEPS
#  include "ChangeEPS_fp.h"
#endif
#if CC_Clear
#  include "Clear_fp.h"
#endif
#if CC_ClearControl
#  include "ClearControl_fp.h"
#endif
#if CC_HierarchyChangeAuth
#  include "HierarchyChangeAuth_fp.h"
#endif
#if CC_DictionaryAttackLockReset
#  include "DictionaryAttackLockReset_fp.h"
#endif
#if CC_DictionaryAttackParameters
#  include "DictionaryAttackParameters_fp.h"
#endif
#if CC_PP_Commands
#  include "PP_Commands_fp.h"
#endif
#if CC_SetAlgorithmSet
#  include "SetAlgorithmSet_fp.h"
#endif
#if CC_FieldUpgradeStart
#  include "FieldUpgradeStart_fp.h"
#endif
#if CC_FieldUpgradeData
#  include "FieldUpgradeData_fp.h"
#endif
#if CC_FirmwareRead
#  include "FirmwareRead_fp.h"
#endif
#if CC_ContextSave
#  include "ContextSave_fp.h"
#endif
#if CC_ContextLoad
#  include "ContextLoad_fp.h"
#endif
#if CC_FlushContext
#  include "FlushContext_fp.h"
#endif
#if CC_EvictControl
#  include "EvictControl_fp.h"
#endif
#if CC_ReadClock
```

```
#  include "ReadClock_fp.h"
#endif
#if CC_ClockSet
#  include "ClockSet_fp.h"
#endif
#if CC_ClockRateAdjust
#  include "ClockRateAdjust_fp.h"
#endif
#if CC_GetCapability
#  include "GetCapability_fp.h"
#endif
#if CC_TestParms
#  include "TestParms_fp.h"
#endif
#if CC_NV_DefineSpace
#  include "NV_DefineSpace_fp.h"
#endif
#if CC_NV_UndefineSpace
#  include "NV_UndefineSpace_fp.h"
#endif
#if CC_NV_UndefineSpaceSpecial
#  include "NV_UndefineSpaceSpecial_fp.h"
#endif
#if CC_NV_ReadPublic
#  include "NV_ReadPublic_fp.h"
#endif
#if CC_NV_Write
#  include "NV_Write_fp.h"
#endif
#if CC_NV_Increment
#  include "NV_Increment_fp.h"
#endif
#if CC_NV_Extend
#  include "NV_Extend_fp.h"
#endif
#if CC_NV_SetBits
#  include "NV_SetBits_fp.h"
#endif
#if CC_NV_WriteLock
#  include "NV_WriteLock_fp.h"
#endif
#if CC_NV_GlobalWriteLock
#  include "NV_GlobalWriteLock_fp.h"
#endif
#if CC_NV_Read
#  include "NV_Read_fp.h"
#endif
#if CC_NV_ReadLock
#  include "NV_ReadLock_fp.h"
#endif
#if CC_NV_ChangeAuth
#  include "NV_ChangeAuth_fp.h"
#endif
#if CC_NV_Certify
#  include "NV_Certify_fp.h"
#endif
#if CC_NV_DefineSpace2
#  include "NV_DefineSpace2_fp.h"
#endif
#if CC_NV_ReadPublic2
#  include "NV_ReadPublic2_fp.h"
#endif
#if CC_SetCapability
#  include "SetCapability_fp.h"
#endif
#if CC_AC_Send
```

```
#  include "AC_Send_fp.h"
#endif
#if CC_Policy_AC_SendSelect
#  include "Policy_AC_SendSelect_fp.h"
#endif
#if CC_ACT_SetTimeout
#  include "ACT_SetTimeout_fp.h"
#endif
#if CC_Vendor_TCG_Test
#  include "Vendor_TCG_Test_fp.h"
#endif

#endif  // _COMMANDS_H_
```

6.16 /tpm/include/private/CryptEcc.h

```
//** Introduction
//
// This file contains structure definitions used for ECC. The structures in this
// file are only used internally. The ECC-related structures that cross the
// public TPM interface are defined in TpmTypes.h
//

// ECC Curve data type decoder ring
// ================================
// | Name                        | Old Name*       | Comments
|
// | ------------------------- | -------------- | ----------------------------------
---------------------------------------------------- |
// | TPM_ECC_CURVE               |                 | 16-bit Curve ID from Part 2 of TCG
TPM Spec                                                  |
// | TPM_ECC_CURVE_METADATA      | ECC_CURVE       | See description below
|
// |                             |                 |
|
// * - if different

// TPM_ECC_CURVE_METADATA
// ======================
// TPM-specific metadata for a particular curve, such as OIDs and signing/kdf
// schemes associated with the curve.
//
// TODO_ECC: Need to remove the curve constants from this structure and replace
// them with a reference to math-lib provided calls. <Once done, add this
// revised comment to the above description> Note: this structure does *NOT*
// include the actual curve constants. The curve constants are no longer in this
// structure because the constants need to be in a format compatible with the
// math library and are retrieved by the `ExtEcc_CurveGet*` family of functions.
//
// Using the math library's constant structure here is not necessary and breaks
// encapsulation.  Using a tpm-specific format means either redundancy (the same
// values exist here and in a math-specific format), or forces the math library
// to adopt a particular format determined by this structure.  Neither outcome
// is as clean as simply leaving the actual constants out of this structure.

#ifndef _CRYPT_ECC_H
#define _CRYPT_ECC_H

//** Structures

#define ECC_BITS (MAX_ECC_KEY_BYTES * 8)
CRYPT_INT_TYPE(ecc, ECC_BITS);

#define CRYPT_ECC_NUM(name) CRYPT_INT_VAR(name, ECC_BITS)
```

```
#define CRYPT_ECC_INITIALIZED(name, initializer) \
    CRYPT_INT_INITIALIZED(name, ECC_BITS, initializer)

typedef struct TPM_ECC_CURVE_METADATA
{
    const TPM_ECC_CURVE   curveId;
    const UINT16          keySizeBits;
    const TPMT_KDF_SCHEME kdf;
    const TPMT_ECC_SCHEME sign;
    const BYTE*           OID;
} TPM_ECC_CURVE_METADATA;

//*** Macros
extern const TPM_ECC_CURVE_METADATA eccCurves[ECC_CURVE_COUNT];

#endif
```

6.17 /tpm/include/private/CryptHash.h

```
//** Introduction
// This header contains the hash structure definitions used in the TPM code
// to define the amount of space to be reserved for the hash state. This allows
// the TPM code to not have to import all of the symbols used by the hash
// computations. This lets the build environment of the TPM code not to have
// include the header files associated with the CryptoEngine code.

#ifndef _CRYPT_HASH_H
#define _CRYPT_HASH_H

//** Hash-related Structures

union SMAC_STATES;

// These definitions add the high-level methods for processing state that may be
// an SMAC
typedef void (*SMAC_DATA_METHOD)(
    union SMAC_STATES* state, UINT32 size, const BYTE* buffer);

typedef UINT16 (*SMAC_END_METHOD)(
    union SMAC_STATES* state, UINT32 size, BYTE* buffer);

typedef struct sequenceMethods
{
    SMAC_DATA_METHOD data;
    SMAC_END_METHOD  end;
} SMAC_METHODS;

#define SMAC_IMPLEMENTED (CC_MAC || CC_MAC_Start)

// These definitions are here because the SMAC state is in the union of hash states.
typedef struct tpmCmacState
{
    TPM_ALG_ID    symAlg;
    UINT16        keySizeBits;
    INT16         bcount;  // current count of bytes accumulated in IV
    TPM2B_IV      iv;      // IV buffer
    TPM2B_SYM_KEY symKey;
} tpmCmacState_t;

typedef union SMAC_STATES
{
#if ALG_CMAC
    tpmCmacState_t cmac;
#endif
    UINT64 pad;
```

```
} SMAC_STATES;

typedef struct SMAC_STATE
{
    SMAC_METHODS smacMethods;
    SMAC_STATES  state;
} SMAC_STATE;

#if ALG_SHA1
#  define IF_IMPLEMENTED_SHA1(op) op(SHA1, Sha1)
#else
#  define IF_IMPLEMENTED_SHA1(op)
#endif
#if ALG_SHA256
#  define IF_IMPLEMENTED_SHA256(op) op(SHA256, Sha256)
#else
#  define IF_IMPLEMENTED_SHA256(op)
#endif
#if ALG_SHA384
#  define IF_IMPLEMENTED_SHA384(op) op(SHA384, Sha384)
#else
#  define IF_IMPLEMENTED_SHA384(op)
#endif
#if ALG_SHA512
#  define IF_IMPLEMENTED_SHA512(op) op(SHA512, Sha512)
#else
#  define IF_IMPLEMENTED_SHA512(op)
#endif
#if ALG_SM3_256
#  define IF_IMPLEMENTED_SM3_256(op) op(SM3_256, Sm3_256)
#else
#  define IF_IMPLEMENTED_SM3_256(op)
#endif
#if ALG_SHA3_256
#  define IF_IMPLEMENTED_SHA3_256(op) op(SHA3_256, Sha3_256)
#else
#  define IF_IMPLEMENTED_SHA3_256(op)
#endif
#if ALG_SHA3_384
#  define IF_IMPLEMENTED_SHA3_384(op) op(SHA3_384, Sha3_384)
#else
#  define IF_IMPLEMENTED_SHA3_384(op)
#endif
#if ALG_SHA3_512
#  define IF_IMPLEMENTED_SHA3_512(op) op(SHA3_512, Sha3_512)
#else
#  define IF_IMPLEMENTED_SHA3_512(op)
#endif

#define FOR_EACH_HASH(op)       \
    IF_IMPLEMENTED_SHA1(op)     \
    IF_IMPLEMENTED_SHA256(op)   \
    IF_IMPLEMENTED_SHA384(op)   \
    IF_IMPLEMENTED_SHA512(op)   \
    IF_IMPLEMENTED_SM3_256(op)  \
    IF_IMPLEMENTED_SHA3_256(op) \
    IF_IMPLEMENTED_SHA3_384(op) \
    IF_IMPLEMENTED_SHA3_512(op)

#define HASH_TYPE(HASH, Hash) tpmHashState##HASH##_t Hash;
typedef union
{
    FOR_EACH_HASH(HASH_TYPE)
// Additions for symmetric block cipher MAC
#if SMAC_IMPLEMENTED
    SMAC_STATE smac;
```

```
#endif
    // to force structure alignment to be no worse than HASH_ALIGNMENT
#if HASH_ALIGNMENT == 8
    uint64_t align;
#else
    uint32_t align;
#endif
} ANY_HASH_STATE;

typedef ANY_HASH_STATE*       PANY_HASH_STATE;
typedef const ANY_HASH_STATE* PCANY_HASH_STATE;

#define ALIGNED_SIZE(x, b) ((((x) + (b)-1) / (b)) * (b))
// MAX_HASH_STATE_SIZE will change with each implementation. It is assumed that
// a hash state will not be larger than twice the block size plus some
// overhead (in this case, 16 bytes). The overall size needs to be as
// large as any of the hash contexts. The structure needs to start on an
// alignment boundary and be an even multiple of the alignment
#define MAX_HASH_STATE_SIZE         ((2 * MAX_HASH_BLOCK_SIZE) + 16)
#define MAX_HASH_STATE_SIZE_ALIGNED ALIGNED_SIZE(MAX_HASH_STATE_SIZE, HASH_ALIGNMENT)

// This is an aligned byte array that will hold any of the hash contexts.
typedef ANY_HASH_STATE ALIGNED_HASH_STATE;

// The header associated with the hash library is expected to define the methods
// which include the calling sequence. When not compiling CryptHash.c, the methods
// are not defined so we need placeholder functions for the structures

#ifndef HASH_START_METHOD_DEF
#  define HASH_START_METHOD_DEF void(HASH_START_METHOD)(void)
#endif
#ifndef HASH_DATA_METHOD_DEF
#  define HASH_DATA_METHOD_DEF void(HASH_DATA_METHOD)(void)
#endif
#ifndef HASH_END_METHOD_DEF
#  define HASH_END_METHOD_DEF void(HASH_END_METHOD)(void)
#endif
#ifndef HASH_STATE_COPY_METHOD_DEF
#  define HASH_STATE_COPY_METHOD_DEF void(HASH_STATE_COPY_METHOD)(void)
#endif
#ifndef HASH_STATE_EXPORT_METHOD_DEF
#  define HASH_STATE_EXPORT_METHOD_DEF void(HASH_STATE_EXPORT_METHOD)(void)
#endif
#ifndef HASH_STATE_IMPORT_METHOD_DEF
#  define HASH_STATE_IMPORT_METHOD_DEF void(HASH_STATE_IMPORT_METHOD)(void)
#endif

// Define the prototypical function call for each of the methods. This defines the
// order in which the parameters are passed to the underlying function.
typedef HASH_START_METHOD_DEF;
typedef HASH_DATA_METHOD_DEF;
typedef HASH_END_METHOD_DEF;
typedef HASH_STATE_COPY_METHOD_DEF;
typedef HASH_STATE_EXPORT_METHOD_DEF;
typedef HASH_STATE_IMPORT_METHOD_DEF;

typedef struct _HASH_METHODS
{
    HASH_START_METHOD*        start;
    HASH_DATA_METHOD*         data;
    HASH_END_METHOD*          end;
    HASH_STATE_COPY_METHOD*   copy;     // Copy a hash block
    HASH_STATE_EXPORT_METHOD* copyOut;  // Copy a hash block from a hash
                                        // context
    HASH_STATE_IMPORT_METHOD* copyIn;   // Copy a hash block to a proper hash
                                        // context
```

```
} HASH_METHODS, *PHASH_METHODS;

#define HASH_TPM2B(HASH, Hash) TPM2B_TYPE(HASH##_DIGEST, HASH##_DIGEST_SIZE);

FOR_EACH_HASH(HASH_TPM2B)

// When the TPM implements RSA, the hash-dependent OID pointers are part of the
// HASH_DEF. These macros conditionally add the OID reference to the HASH_DEF and the
// HASH_DEF_TEMPLATE.
#if ALG_RSA
#  define PKCS1_HASH_REF  const BYTE* PKCS1;
#  define PKCS1_OID(NAME) , OID_PKCS1_##NAME
#else
#  define PKCS1_HASH_REF
#  define PKCS1_OID(NAME)
#endif

// When the TPM implements ECC, the hash-dependent OID pointers are part of the
// HASH_DEF. These macros conditionally add the OID reference to the HASH_DEF and the
// HASH_DEF_TEMPLATE.
#if ALG_ECDSA
#  define ECDSA_HASH_REF  const BYTE* ECDSA;
#  define ECDSA_OID(NAME) , OID_ECDSA_##NAME
#else
#  define ECDSA_HASH_REF
#  define ECDSA_OID(NAME)
#endif

typedef const struct HASH_DEF_STRUCT
{
    HASH_METHODS method;
    uint16_t     blockSize;
    uint16_t     digestSize;
    uint16_t     contextSize;
    uint16_t     hashAlg;
    const BYTE*  OID;
    PKCS1_HASH_REF      // PKCS1 OID
        ECDSA_HASH_REF  // ECDSA OID
} HASH_DEF, *PHASH_DEF;

// Macro to fill in the HASH_DEF for an algorithm. For SHA1, the instance would be:
//  HASH_DEF_TEMPLATE(Sha1, SHA1)
// This handles the difference in capitalization for the various pieces.
#define HASH_DEF_TEMPLATE(HASH, Hash)                                       \
    HASH_DEF Hash##_Def =                                                   \
        {{                                                                  \
             (HASH_START_METHOD*)&tpmHashStart_##HASH,                      \
             (HASH_DATA_METHOD*)&tpmHashData_##HASH,                        \
             (HASH_END_METHOD*)&tpmHashEnd_##HASH,                          \
             (HASH_STATE_COPY_METHOD*)&tpmHashStateCopy_##HASH,             \
             (HASH_STATE_EXPORT_METHOD*)&tpmHashStateExport_##HASH,         \
             (HASH_STATE_IMPORT_METHOD*)&tpmHashStateImport_##HASH,         \
         },                                                                 \
         HASH##_BLOCK_SIZE,  /*block size */                                \
         HASH##_DIGEST_SIZE, /*data size */                                 \
         sizeof(tpmHashState##HASH##_t),                                    \
         TPM_ALG_##HASH,                                                    \
         OID_##HASH PKCS1_OID(HASH) ECDSA_OID(HASH)};

// These definitions are for the types that can be in a hash state structure.
// These types are used in the cryptographic utilities. This is a define rather than
// an enum so that the size of this field can be explicit.
typedef BYTE HASH_STATE_TYPE;
#define HASH_STATE_EMPTY ((HASH_STATE_TYPE)0)
#define HASH_STATE_HASH  ((HASH_STATE_TYPE)1)
#define HASH_STATE_HMAC  ((HASH_STATE_TYPE)2)
```

```
#if CC_MAC || CC_MAC_Start
#  define HASH_STATE_SMAC ((HASH_STATE_TYPE)3)
#endif

// This is the structure that is used for passing a context into the hashing
// functions. It should be the same size as the function context used within
// the hashing functions. This is checked when the hash function is initialized.
// This version uses a new layout for the contexts and a different definition. The
// state buffer is an array of HASH_UNIT values so that a decent compiler will put
// the structure on a HASH_UNIT boundary. If the structure is not properly aligned,
// the code that manipulates the structure will copy to a properly aligned
// structure before it is used and copy the result back. This just makes things
// slower.
// NOTE: This version of the state had the pointer to the update method in the
// state. This is to allow the SMAC functions to use the same structure without
// having to replicate the entire HASH_DEF structure.
typedef struct _HASH_STATE
{
    HASH_STATE_TYPE type;  // type of the context
    TPM_ALG_ID      hashAlg;
    PHASH_DEF       def;
    ANY_HASH_STATE  state;
} HASH_STATE, *PHASH_STATE;
typedef const HASH_STATE* PCHASH_STATE;

//** HMAC State Structures

// An HMAC_STATE structure contains an opaque HMAC stack state. A caller would
// use this structure when performing incremental HMAC operations. This structure
// contains a hash state and an HMAC key and allows slightly better stack
// optimization than adding an HMAC key to each hash state.
typedef struct hmacState
{
    HASH_STATE       hashState;  // the hash state
    TPM2B_HASH_BLOCK hmacKey;    // the HMAC key
} HMAC_STATE, *PHMAC_STATE;

// This is for the external hash state. This implementation assumes that the size
// of the exported hash state is no larger than the internal hash state.
typedef struct
{
    BYTE buffer[sizeof(HASH_STATE)];
} EXPORT_HASH_STATE, *PEXPORT_HASH_STATE;

typedef const EXPORT_HASH_STATE* PCEXPORT_HASH_STATE;

#endif  //  _CRYPT_HASH_H
```

6.18 /tpm/include/private/CryptRand.h

```
//** Introduction
// This file contains constant definition shared by CryptUtil and the parts
// of the Crypto Engine.
//

#ifndef _CRYPT_RAND_H
#define _CRYPT_RAND_H

//** DRBG Structures and Defines

// Values and structures for the random number generator. These values are defined
// in this header file so that the size of the RNG state can be known to TPM.lib.
// This allows the allocation of some space in NV memory for the state to
// be stored on an orderly shutdown.
```

```
// The DRBG based on a symmetric block cipher is defined by three values,
// 1) the key size
// 2) the block size (the IV size)
// 3) the symmetric algorithm

#define DRBG_KEY_SIZE_BITS AES_MAX_KEY_SIZE_BITS
#define DRBG_IV_SIZE_BITS  (AES_MAX_BLOCK_SIZE * 8)
#define DRBG_ALGORITHM     TPM_ALG_AES

#define DRBG_ENCRYPT_SETUP(key, keySizeInBits, schedule) \
    TpmCryptSetEncryptKeyAES(key, keySizeInBits, schedule)
#define DRBG_ENCRYPT(keySchedule, in, out) \
    TpmCryptEncryptAES(SWIZZLE(keySchedule, in, out))

#if((DRBG_KEY_SIZE_BITS % RADIX_BITS) != 0) || ((DRBG_IV_SIZE_BITS % RADIX_BITS) != 0)
#  error "Key size and IV for DRBG must be even multiples of the radix"
#endif
#if(DRBG_KEY_SIZE_BITS % DRBG_IV_SIZE_BITS) != 0
#  error "Key size for DRBG must be even multiple of the cypher block size"
#endif

// Derived values
#define DRBG_MAX_REQUESTS_PER_RESEED (1 << 48)
#define DRBG_MAX_REQEST_SIZE         (1 << 32)

#define pDRBG_KEY(seed) ((DRBG_KEY*)&(((BYTE*)(seed))[0]))
#define pDRBG_IV(seed)  ((DRBG_IV*)&(((BYTE*)(seed))[DRBG_KEY_SIZE_BYTES]))

#define DRBG_KEY_SIZE_WORDS (BITS_TO_CRYPT_WORDS(DRBG_KEY_SIZE_BITS))
#define DRBG_KEY_SIZE_BYTES (DRBG_KEY_SIZE_WORDS * RADIX_BYTES)

#define DRBG_IV_SIZE_WORDS (BITS_TO_CRYPT_WORDS(DRBG_IV_SIZE_BITS))
#define DRBG_IV_SIZE_BYTES (DRBG_IV_SIZE_WORDS * RADIX_BYTES)

#define DRBG_SEED_SIZE_WORDS (DRBG_KEY_SIZE_WORDS + DRBG_IV_SIZE_WORDS)
#define DRBG_SEED_SIZE_BYTES (DRBG_KEY_SIZE_BYTES + DRBG_IV_SIZE_BYTES)

typedef union
{
    BYTE          bytes[DRBG_KEY_SIZE_BYTES];
    crypt_uword_t words[DRBG_KEY_SIZE_WORDS];
} DRBG_KEY;

typedef union
{
    BYTE          bytes[DRBG_IV_SIZE_BYTES];
    crypt_uword_t words[DRBG_IV_SIZE_WORDS];
} DRBG_IV;

typedef union
{
    BYTE          bytes[DRBG_SEED_SIZE_BYTES];
    crypt_uword_t words[DRBG_SEED_SIZE_WORDS];
} DRBG_SEED;

#define CTR_DRBG_MAX_REQUESTS_PER_RESEED ((UINT64)1 << 20)
#define CTR_DRBG_MAX_BYTES_PER_REQUEST   (1 << 16)

#define CTR_DRBG_MIN_ENTROPY_INPUT_LENGTH    DRBG_SEED_SIZE_BYTES
#define CTR_DRBG_MAX_ENTROPY_INPUT_LENGTH    DRBG_SEED_SIZE_BYTES
#define CTR_DRBG_MAX_ADDITIONAL_INPUT_LENGTH DRBG_SEED_SIZE_BYTES

#define TESTING (1 << 0)
#define ENTROPY (1 << 1)
#define TESTED  (1 << 2)
```

```
#define IsTestStateSet(BIT)    ((g_cryptoSelfTestState.rng & BIT) != 0)
#define SetTestStateBit(BIT)   (g_cryptoSelfTestState.rng |= BIT)
#define ClearTestStateBit(BIT) (g_cryptoSelfTestState.rng &= ~BIT)

#define IsSelfTest()    IsTestStateSet(TESTING)
#define SetSelfTest()   SetTestStateBit(TESTING)
#define ClearSelfTest() ClearTestStateBit(TESTING)

#define IsEntropyBad()    IsTestStateSet(ENTROPY)
#define SetEntropyBad()   SetTestStateBit(ENTROPY)
#define ClearEntropyBad() ClearTestStateBit(ENTROPY)

#define IsDrbgTested()    IsTestStateSet(TESTED)
#define SetDrbgTested()   SetTestStateBit(TESTED)
#define ClearDrbgTested() ClearTestStateBit(TESTED)

typedef struct
{
    UINT64    reseedCounter;
    UINT32    magic;
    DRBG_SEED seed;          // contains the key and IV for the counter mode DRBG
    UINT32    lastValue[4];  // used when the TPM does continuous self-test
                             // for FIPS compliance of DRBG
} DRBG_STATE, *pDRBG_STATE;
#define DRBG_MAGIC ((UINT32)0x47425244)  // "DRBG" backwards so that it displays

typedef struct KDF_STATE
{
    UINT64       counter;
    UINT32       magic;
    UINT32       limit;
    TPM2B*       seed;
    const TPM2B* label;
    TPM2B*       context;
    TPM_ALG_ID   hash;
    TPM_ALG_ID   kdf;
    UINT16       digestSize;
    TPM2B_DIGEST residual;
} KDF_STATE, *pKDR_STATE;
#define KDF_MAGIC ((UINT32)0x4048444a)  // "KDF " backwards

// Make sure that any other structures added to this union start with a 64-bit
// counter and a 32-bit magic number
typedef union
{
    DRBG_STATE drbg;
    KDF_STATE  kdf;
} RAND_STATE;

// This is the state used when the library uses a random number generator.
// A special function is installed for the library to call. That function
// picks up the state from this location and uses it for the generation
// of the random number.
extern RAND_STATE* s_random;

// When instrumenting RSA key sieve
#if RSA_INSTRUMENT
#  define PRIME_INDEX(x)       ((x) == 512 ? 0 : (x) == 1024 ? 1 : 2)
#  define INSTRUMENT_SET(a, b) ((a) = (b))
#  define INSTRUMENT_ADD(a, b) (a) = (a) + (b)
#  define INSTRUMENT_INC(a)    (a) = (a) + 1

extern UINT32 PrimeIndex;
extern UINT32 failedAtIteration[10];
extern UINT32 PrimeCounts[3];
extern UINT32 MillerRabinTrials[3];
```

```
extern UINT32 totalFieldsSieved[3];
extern UINT32 bitsInFieldAfterSieve[3];
extern UINT32 emptyFieldsSieved[3];
extern UINT32 noPrimeFields[3];
extern UINT32 primesChecked[3];
extern UINT16 lastSievePrime;
#else
#  define INSTRUMENT_SET(a, b)
#  define INSTRUMENT_ADD(a, b)
#  define INSTRUMENT_INC(a)
#endif

#endif  // _CRYPT_RAND_H
```

6.19 /tpm/include/private/CryptRsa.h

```
// This file contains the RSA-related structures and defines.

#ifndef _CRYPT_RSA_H
#define _CRYPT_RSA_H

// These values are used in the Crypt_Int* representation of various RSA values.
// define ci_rsa_t as buffer containing a CRYPT_INT object with space for
// (MAX_RSA_KEY_BITS) of actual data.
CRYPT_INT_TYPE(rsa, MAX_RSA_KEY_BITS);
#define CRYPT_RSA_VAR(name) CRYPT_INT_VAR(name, MAX_RSA_KEY_BITS)
#define CRYPT_RSA_INITIALIZED(name, initializer) \
    CRYPT_INT_INITIALIZED(name, MAX_RSA_KEY_BITS, initializer)

#define CRYPT_PRIME_VAR(name) CRYPT_INT_VAR(name, (MAX_RSA_KEY_BITS / 2))
// define ci_prime_t as buffer containing a CRYPT_INT object with space for
// (MAX_RSA_KEY_BITS/2) of actual data.
CRYPT_INT_TYPE(prime, (MAX_RSA_KEY_BITS / 2));
#define CRYPT_PRIME_INITIALIZED(name, initializer) \
    CRYPT_INT_INITIALIZED(name, MAX_RSA_KEY_BITS / 2, initializer)

#if !CRT_FORMAT_RSA
#  error This verson only works with CRT formatted data
#endif  // !CRT_FORMAT_RSA

typedef struct privateExponent
{
    Crypt_Int* P;
    Crypt_Int* Q;
    Crypt_Int* dP;
    Crypt_Int* dQ;
    Crypt_Int* qInv;
    ci_prime_t entries[5];
} privateExponent;

#define NEW_PRIVATE_EXPONENT(X) \
    privateExponent  _##X;      \
    privateExponent* X = RsaInitializeExponent(&(_##X))

#endif  // _CRYPT_RSA_H
```

6.20 /tpm/include/private/CryptSym.h

```
//** Introduction
//
// This file contains the implementation of the symmetric block cipher modes
// allowed for a TPM. These functions only use the single block encryption functions
// of the selected symmetric cryptographic library.
```

```
//** Includes, Defines, and Typedefs
#ifndef CRYPT_SYM_H
#define CRYPT_SYM_H

#if ALG_AES
#  define IF_IMPLEMENTED_AES(op) op(AES, aes)
#else
#  define IF_IMPLEMENTED_AES(op)
#endif
#if ALG_SM4
#  define IF_IMPLEMENTED_SM4(op) op(SM4, sm4)
#else
#  define IF_IMPLEMENTED_SM4(op)
#endif
#if ALG_CAMELLIA
#  define IF_IMPLEMENTED_CAMELLIA(op) op(CAMELLIA, camellia)
#else
#  define IF_IMPLEMENTED_CAMELLIA(op)
#endif

#define FOR_EACH_SYM(op)    \
    IF_IMPLEMENTED_AES(op) \
    IF_IMPLEMENTED_SM4(op) \
    IF_IMPLEMENTED_CAMELLIA(op)

// Macros for creating the key schedule union
#define KEY_SCHEDULE(SYM, sym) tpmKeySchedule##SYM sym;
typedef union tpmCryptKeySchedule_t
{
    FOR_EACH_SYM(KEY_SCHEDULE)

#if SYMMETRIC_ALIGNMENT == 8
    uint64_t alignment;
#else
    uint32_t alignment;
#endif
} tpmCryptKeySchedule_t;

// Each block cipher within a library is expected to conform to the same calling
// conventions with three parameters ('keySchedule', 'in', and 'out') in the same
// order. That means that all algorithms would use the same order of the same
// parameters. The code is written assuming the ('keySchedule', 'in', and 'out')
// order. However, if the library uses a different order, the order can be changed
// with a SWIZZLE macro that puts the parameters in the correct order.
// Note that all algorithms have to use the same order and number of parameters
// because the code to build the calling list is common for each call to encrypt
// or decrypt with the algorithm chosen by setting a function pointer to select
// the algorithm that is used.

#define ENCRYPT(keySchedule, in, out) encrypt(SWIZZLE(keySchedule, in, out))

#define DECRYPT(keySchedule, in, out) decrypt(SWIZZLE(keySchedule, in, out))

// Note that the macros rely on 'encrypt' as local values in the
// functions that use these macros. Those parameters are set by the macro that
// set the key schedule to be used for the call.

#define ENCRYPT_CASE(ALG, alg)                                              \
    case TPM_ALG_##ALG:                                                     \
        TpmCryptSetEncryptKey##ALG(key, keySizeInBits, &keySchedule.alg); \
        encrypt = (TpmCryptSetSymKeyCall_t)TpmCryptEncrypt##ALG;           \
        break;
#define DECRYPT_CASE(ALG, alg)                                              \
    case TPM_ALG_##ALG:                                                     \
        TpmCryptSetDecryptKey##ALG(key, keySizeInBits, &keySchedule.alg); \
        decrypt = (TpmCryptSetSymKeyCall_t)TpmCryptDecrypt##ALG;           \
```

```
        break;

#endif  // CRYPT_SYM_H
```

6.21 /tpm/include/private/CryptTest.h

```
// This file contains constant definitions used for self-test.

#ifndef _CRYPT_TEST_H
#define _CRYPT_TEST_H

// This is the definition of a bit array with one bit per algorithm.
// NOTE: Since bit numbering starts at zero, when TPM_ALG_LAST is a multiple of 8,
// ALGORITHM_VECTOR will need to have byte for the single bit in the last byte. So,
// for example, when TPM_ALG_LAST is 8, ALGORITHM_VECTOR will need 2 bytes.
#define ALGORITHM_VECTOR_BYTES ((TPM_ALG_LAST + 8) / 8)
typedef BYTE ALGORITHM_VECTOR[ALGORITHM_VECTOR_BYTES];

#ifdef TEST_SELF_TEST
LIB_EXPORT extern ALGORITHM_VECTOR LibToTest;
#endif

// This structure is used to contain self-test tracking information for the
// cryptographic modules. Each of the major modules is given a 32-bit value in
// which it may maintain its own self test information. The convention for this
// state is that when all of the bits in this structure are 0, all functions need
// to be tested.
typedef struct
{
    UINT32 rng;
    UINT32 hash;
    UINT32 sym;
#if ALG_RSA
    UINT32 rsa;
#endif
#if ALG_ECC
    UINT32 ecc;
#endif
} CRYPTO_SELF_TEST_STATE;

#endif  // _CRYPT_TEST_H
```

6.22 /tpm/include/private/EccTestData.h

```
// This file contains the parameter data for ECC testing.

#ifdef SELF_TEST_DATA

TPM2B_TYPE(EC_TEST, 32);
const TPM_ECC_CURVE c_testCurve = 00003;

// The "static" key

const TPM2B_EC_TEST c_ecTestKey_ds = {
    {32, {0xdf, 0x8d, 0xa4, 0xa3, 0x88, 0xf6, 0x76, 0x96, 0x89, 0xfc, 0x2f,
          0x2d, 0xa1, 0xb4, 0x39, 0x7a, 0x78, 0xc4, 0x7f, 0x71, 0x8c, 0xa6,
          0x91, 0x85, 0xc0, 0xbf, 0xf3, 0x54, 0x20, 0x91, 0x2f, 0x73}}};

const TPM2B_EC_TEST c_ecTestKey_QsX = {
    {32, {0x17, 0xad, 0x2f, 0xcb, 0x18, 0xd4, 0xdb, 0x3f, 0x2c, 0x53, 0x13,
          0x82, 0x42, 0x97, 0xff, 0x8d, 0x99, 0x50, 0x16, 0x02, 0x35, 0xa7,
          0x06, 0xae, 0x1f, 0xda, 0xe2, 0x9c, 0x12, 0x77, 0xc0, 0xf9}}};

const TPM2B_EC_TEST c_ecTestKey_QsY = {
```

```
    {32, {0xa6, 0xca, 0xf2, 0x18, 0x45, 0x96, 0x6e, 0x58, 0xe6, 0x72, 0x34,
          0x12, 0x89, 0xcd, 0xaa, 0xad, 0xcb, 0x68, 0xb2, 0x51, 0xdc, 0x5e,
          0xd1, 0x6d, 0x38, 0x20, 0x35, 0x57, 0xb2, 0xfd, 0xc7, 0x52}}};

// The "ephemeral" key

const TPM2B_EC_TEST c_ecTestKey_de = {
    {32, {0xb6, 0xb5, 0x33, 0x5c, 0xd1, 0xee, 0x52, 0x07, 0x99, 0xea, 0x2e,
          0x8f, 0x8b, 0x19, 0x18, 0x07, 0xc1, 0xf8, 0xdf, 0xdd, 0xb8, 0x77,
          0x00, 0xc7, 0xd6, 0x53, 0x21, 0xed, 0x02, 0x53, 0xee, 0xac}}};

const TPM2B_EC_TEST c_ecTestKey_QeX = {
    {32, {0xa5, 0x1e, 0x80, 0xd1, 0x76, 0x3e, 0x8b, 0x96, 0xce, 0xcc, 0x21,
          0x82, 0xc9, 0xa2, 0xa2, 0xed, 0x47, 0x21, 0x89, 0x53, 0x44, 0xe9,
          0xc7, 0x92, 0xe7, 0x31, 0x48, 0x38, 0xe6, 0xea, 0x93, 0x47}}};

const TPM2B_EC_TEST c_ecTestKey_QeY = {
    {32, {0x30, 0xe6, 0x4f, 0x97, 0x03, 0xa1, 0xcb, 0x3b, 0x32, 0x2a, 0x70,
          0x39, 0x94, 0xeb, 0x4e, 0xea, 0x55, 0x88, 0x81, 0x3f, 0xb5, 0x00,
          0xb8, 0x54, 0x25, 0xab, 0xd4, 0xda, 0xfd, 0x53, 0x7a, 0x18}}};

// ECDH test results
const TPM2B_EC_TEST c_ecTestEcdh_X = {
    {32, {0x64, 0x02, 0x68, 0x92, 0x78, 0xdb, 0x33, 0x52, 0xed, 0x3b, 0xfa,
          0x3b, 0x74, 0xa3, 0x3d, 0x2c, 0x2f, 0x9c, 0x59, 0x03, 0x07, 0xf8,
          0x22, 0x90, 0xed, 0xe3, 0x45, 0xf8, 0x2a, 0x0a, 0xd8, 0x1d}}};

const TPM2B_EC_TEST c_ecTestEcdh_Y = {
    {32, {0x58, 0x94, 0x05, 0x82, 0xbe, 0x5f, 0x33, 0x02, 0x25, 0x90, 0x3a,
          0x33, 0x90, 0x89, 0xe3, 0xe5, 0x10, 0x4a, 0xbc, 0x78, 0xa5, 0xc5,
          0x07, 0x64, 0xaf, 0x91, 0xbc, 0xe6, 0xff, 0x85, 0x11, 0x40}}};

TPM2B_TYPE(TEST_VALUE, 64);
const TPM2B_TEST_VALUE c_ecTestValue = {
    {64,
     {0x78, 0xd5, 0xd4, 0x56, 0x43, 0x61, 0xdb, 0x97, 0xa4, 0x32, 0xc4, 0x0b, 0x06,
      0xa9, 0xa8, 0xa0, 0xf4, 0x45, 0x7f, 0x13, 0xd8, 0x13, 0x81, 0x0b, 0xe5, 0x76,
      0xbe, 0xaa, 0xb6, 0x3f, 0x8d, 0x4d, 0x23, 0x65, 0xcc, 0xa7, 0xc9, 0x19, 0x10,
      0xce, 0x69, 0xcb, 0x0c, 0xc7, 0x11, 0x8d, 0xc3, 0xff, 0x62, 0x69, 0xa2, 0xbe,
      0x46, 0x90, 0xe7, 0x7d, 0x81, 0x77, 0x94, 0x65, 0x1c, 0x3e, 0xc1, 0x3e}}};

#  if ALG_SHA1_VALUE == DEFAULT_TEST_HASH

const TPM2B_EC_TEST c_TestEcDsa_r = {
    {32, {0x57, 0xf3, 0x36, 0xb7, 0xec, 0xc2, 0xdd, 0x76, 0x0e, 0xe2, 0x81,
          0x21, 0x49, 0xc5, 0x66, 0x11, 0x4b, 0x8a, 0x4f, 0x17, 0x62, 0x82,
          0xcc, 0x06, 0xf6, 0x64, 0x78, 0xef, 0x6b, 0x7c, 0xf2, 0x6c}}};
const TPM2B_EC_TEST c_TestEcDsa_s = {
    {32, {0x1b, 0xed, 0x23, 0x72, 0x8f, 0x17, 0x5f, 0x47, 0x2e, 0xa7, 0x97,
          0x2c, 0x51, 0x57, 0x20, 0x70, 0x6f, 0x89, 0x74, 0x8a, 0xa8, 0xf4,
          0x26, 0xf4, 0x96, 0xa1, 0xb8, 0x3e, 0xe5, 0x35, 0xc5, 0x94}}};

const TPM2B_EC_TEST c_TestEcSchnorr_r = {
    {32, {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x1b, 0x08, 0x9f, 0xde, 0xef, 0x62, 0xe3, 0xf1, 0x14, 0xcb,
          0x54, 0x28, 0x13, 0x76, 0xfc, 0x6d, 0x69, 0x22, 0xb5, 0x3e}}};
const TPM2B_EC_TEST c_TestEcSchnorr_s = {
    {32, {0xd9, 0xd3, 0x20, 0xfb, 0x4d, 0x16, 0xf2, 0xe6, 0xe2, 0x45, 0x07,
          0x45, 0x1c, 0x92, 0x92, 0x92, 0xa9, 0x6b, 0x48, 0xf8, 0xd1, 0x98,
          0x29, 0x4d, 0xd3, 0x8f, 0x56, 0xf2, 0xbb, 0x2e, 0x22, 0x3b}}};

#  endif  // SHA1

#  if ALG_SHA256_VALUE == DEFAULT_TEST_HASH

const TPM2B_EC_TEST c_TestEcDsa_r = {
```

```
    {32, {0x04, 0x7d, 0x54, 0xeb, 0x04, 0x6f, 0x56, 0xec, 0xa2, 0x6c, 0x38,
          0x8c, 0xeb, 0x43, 0x0b, 0x71, 0xf8, 0xf2, 0xf4, 0xa5, 0xe0, 0x1d,
          0x3c, 0xa2, 0x39, 0x31, 0xe4, 0xe7, 0x36, 0x3b, 0xb5, 0x5f}}};
const TPM2B_EC_TEST c_TestEcDsa_s = {
    {32, {0x8f, 0xd0, 0x12, 0xd9, 0x24, 0x75, 0xf6, 0xc4, 0x3b, 0xb5, 0x46,
          0x75, 0x3a, 0x41, 0x8d, 0x80, 0x23, 0x99, 0x38, 0xd7, 0xe2, 0x40,
          0xca, 0x9a, 0x19, 0x2a, 0xfc, 0x54, 0x75, 0xd3, 0x4a, 0x6e}}};

const TPM2B_EC_TEST c_TestEcSchnorr_r = {
    {32, {0xf7, 0xb9, 0x15, 0x4c, 0x34, 0xf6, 0x41, 0x19, 0xa3, 0xd2, 0xf1,
          0xbd, 0xf4, 0x13, 0x6a, 0x4f, 0x63, 0xb8, 0x4d, 0xb5, 0xc8, 0xcd,
          0xde, 0x85, 0x95, 0xa5, 0x39, 0x0a, 0x14, 0x49, 0x3d, 0x2f}}};
const TPM2B_EC_TEST c_TestEcSchnorr_s = {
    {32, {0xfe, 0xbe, 0x17, 0xaa, 0x31, 0x22, 0x9f, 0xd0, 0xd2, 0xf5, 0x25,
          0x04, 0x92, 0xb0, 0xaa, 0x4e, 0xcc, 0x1c, 0xb6, 0x79, 0xd6, 0x42,
          0xb3, 0x4e, 0x3f, 0xbb, 0xfe, 0x5f, 0xd0, 0xd0, 0x8b, 0xc3}}};

#  endif  // SHA256

#  if ALG_SHA384_VALUE == DEFAULT_TEST_HASH

const TPM2B_EC_TEST c_TestEcDsa_r = {
    {32, {0xf5, 0x74, 0x6d, 0xd6, 0xc6, 0x56, 0x86, 0xbb, 0xba, 0x1c, 0xba,
          0x75, 0x65, 0xee, 0x64, 0x31, 0xce, 0x04, 0xe3, 0x9f, 0x24, 0x3f,
          0xbd, 0xfe, 0x04, 0xcd, 0xab, 0x7e, 0xfe, 0xad, 0xcb, 0x82}}};
const TPM2B_EC_TEST c_TestEcDsa_s = {
    {32, {0xc2, 0x4f, 0x32, 0xa1, 0x06, 0xc0, 0x85, 0x4f, 0xc6, 0xd8, 0x31,
          0x66, 0x91, 0x9f, 0x79, 0xcd, 0x5b, 0xe5, 0x7b, 0x94, 0xa1, 0x91,
          0x38, 0xac, 0xd4, 0x20, 0xa2, 0x10, 0xf0, 0xd5, 0x9d, 0xbf}}};

const TPM2B_EC_TEST c_TestEcSchnorr_r = {
    {32, {0x1e, 0xb8, 0xe1, 0xbf, 0xa1, 0x9e, 0x39, 0x1e, 0x58, 0xa2, 0xe6,
          0x59, 0xd0, 0x1a, 0x6a, 0x03, 0x6a, 0x1f, 0x1c, 0x4f, 0x36, 0x19,
          0xc1, 0xec, 0x30, 0xa4, 0x85, 0x1b, 0xe9, 0x74, 0x35, 0x66}}};
const TPM2B_EC_TEST c_TestEcSchnorr_s = {
    {32, {0xb9, 0xe6, 0xe3, 0x7e, 0xcb, 0xb9, 0xea, 0xf1, 0xcc, 0xf4, 0x48,
          0x44, 0x4a, 0xda, 0xc8, 0xd7, 0x87, 0xb4, 0xba, 0x40, 0xfe, 0x5b,
          0x68, 0x11, 0x14, 0xcf, 0xa0, 0x0e, 0x85, 0x46, 0x99, 0x01}}};

#  endif  // SHA384

#  if ALG_SHA512_VALUE == DEFAULT_TEST_HASH

const TPM2B_EC_TEST c_TestEcDsa_r = {
    {32, {0xc9, 0x71, 0xa6, 0xb4, 0xaf, 0x46, 0x26, 0x8c, 0x27, 0x00, 0x06,
          0x3b, 0x00, 0x0f, 0xa3, 0x17, 0x72, 0x48, 0x40, 0x49, 0x4d, 0x51,
          0x4f, 0xa4, 0xcb, 0x7e, 0x86, 0xe9, 0xe7, 0xb4, 0x79, 0xb2}}};
const TPM2B_EC_TEST c_TestEcDsa_s = {
    {32, {0x87, 0xbc, 0xc0, 0xed, 0x74, 0x60, 0x9e, 0xfa, 0x4e, 0xe8, 0x16,
          0xf3, 0xf9, 0x6b, 0x26, 0x07, 0x3c, 0x74, 0x31, 0x7e, 0xf0, 0x62,
          0x46, 0xdc, 0xd6, 0x45, 0x22, 0x47, 0x3e, 0x0c, 0xa0, 0x02}}};

const TPM2B_EC_TEST c_TestEcSchnorr_r = {
    {32, {0xcc, 0x07, 0xad, 0x65, 0x91, 0xdd, 0xa0, 0x10, 0x23, 0xae, 0x53,
          0xec, 0xdf, 0xf1, 0x50, 0x90, 0x16, 0x96, 0xf4, 0x45, 0x09, 0x73,
          0x9c, 0x84, 0xb5, 0x5c, 0x5f, 0x08, 0x51, 0xcb, 0x60, 0x01}}};
const TPM2B_EC_TEST c_TestEcSchnorr_s = {
    {32, {0x55, 0x20, 0x21, 0x54, 0xe2, 0x49, 0x07, 0x47, 0x71, 0xf4, 0x99,
          0x15, 0x54, 0xf3, 0xab, 0x14, 0xdb, 0x8e, 0xda, 0x79, 0xb6, 0x02,
          0x0e, 0xe3, 0x5e, 0x6f, 0x2c, 0xb6, 0x05, 0xbd, 0x14, 0x10}}};

#  endif  // SHA512

#endif  // SELF_TEST_DATA
```

6.23 /tpm/include/private/Global.h

```
//** Description

// This file contains internal global type definitions and data declarations that
// are need between subsystems. The instantiation of global data is in Global.c.
// The initialization of global data is in the subsystem that is the primary owner
// of the data.
//
// The first part of this file has the 'typedefs' for structures and other defines
// used in many portions of the code. After the 'typedef' section, is a section that
// defines global values that are only present in RAM. The next three sections
// define the structures for the NV data areas: persistent, orderly, and state
// save. Additional sections define the data that is used in specific modules. That
// data is private to the module but is collected here to simplify the management
// of the instance data.
//
// All the data is instanced in Global.c.
#if !defined _TPM_H_
#  error "Should only be instanced in TPM.h"
#endif

//** Includes

#ifndef GLOBAL_H
#  define GLOBAL_H

_REDUCE_WARNING_LEVEL_(2)
#  include <string.h>
#  include <stddef.h>
_NORMAL_WARNING_LEVEL_

#  include "GpMacros.h"
#  include "Capabilities.h"
#  include "TpmTypes.h"  // requires GpMacros & Capabilities
#  include "CommandAttributes.h"
#  include "CryptTest.h"

#  ifndef MATH_LIB
#    error MATH_LIB required
#  endif
#  include LIB_INCLUDE(TpmTo, MATH_LIB, Math)

#  include "CryptHash.h"
#  include "CryptSym.h"
#  include "CryptRand.h"
#  include "CryptEcc.h"
#  include "CryptRsa.h"
#  include "CryptTest.h"
#  include "NV.h"
#  include "ACT.h"

//** Defines and Types

//*** Other Types
// An AUTH_VALUE is a BYTE array containing a digest (TPMU_HA)
typedef BYTE AUTH_VALUE[sizeof(TPMU_HA)];

// A TIME_INFO is a BYTE array that can contain a TPMS_TIME_INFO
typedef BYTE TIME_INFO[sizeof(TPMS_TIME_INFO)];

// A NAME is a BYTE array that can contain a TPMU_NAME
typedef BYTE NAME[sizeof(TPMU_NAME)];

// Definition for a PROOF value
TPM2B_TYPE(PROOF, PROOF_SIZE);
```

```
// Definition for a Primary Seed value
TPM2B_TYPE(SEED, PRIMARY_SEED_SIZE);

// A CLOCK_NONCE is used to tag the time value in the authorization session and
// in the ticket computation so that the ticket expires when there is a time
// discontinuity. When the clock stops during normal operation, the nonce is
// 64-bit value kept in RAM but it is a 32-bit counter when the clock only stops
// during power events.
#  if CLOCK_STOPS
typedef UINT64 CLOCK_NONCE;
#  else
typedef UINT32 CLOCK_NONCE;
#  endif

//** Loaded Object Structures
//*** Description
// The structures in this section define the object layout as it exists in TPM
// memory.
//
// Two types of objects are defined: an ordinary object such as a key, and a
// sequence object that may be a hash, HMAC, or event.
//
//*** OBJECT_ATTRIBUTES
// An OBJECT_ATTRIBUTES structure contains the variable attributes of an object.
// These properties are not part of the public properties but are used by the
// TPM in managing the object. An OBJECT_ATTRIBUTES is used in the definition of
// the OBJECT data type.

typedef struct
{
    unsigned publicOnly : 1;    //0) SET if only the public portion of
                                //   an object is loaded
    unsigned epsHierarchy : 1;  //1) SET if the object belongs to EPS
                                //   Hierarchy
    unsigned ppsHierarchy : 1;  //2) SET if the object belongs to PPS
                                //   Hierarchy
    unsigned spsHierarchy : 1;  //3) SET f the object belongs to SPS
                                //   Hierarchy
    unsigned evict : 1;         //4) SET if the object is a platform or
                                //   owner evict object.  Platform-
                                //   evict object belongs to PPS
                                //   hierarchy, owner-evict object
                                //   belongs to SPS or EPS hierarchy.
                                //   This bit is also used to mark a
                                //   completed sequence object so it
                                //   will be flush when the
                                //   SequenceComplete command succeeds.
    unsigned primary   : 1;     //5) SET for a primary object
    unsigned temporary : 1;     //6) SET for a temporary object
    unsigned stClear   : 1;     //7) SET for an stClear object
    unsigned hmacSeq   : 1;     //8) SET for an HMAC or MAC sequence
                                //   object
    unsigned hashSeq    : 1;    //9) SET for a hash sequence object
    unsigned eventSeq   : 1;    //10) SET for an event sequence object
    unsigned ticketSafe : 1;    //11) SET if a ticket is safe to create
                                //    for hash sequence object
    unsigned firstBlock : 1;    //12) SET if the first block of hash
                                //    data has been received.  It
                                //    works with ticketSafe bit
    unsigned isParent : 1;      //13) SET if the key has the proper
                                //    attributes to be a parent key
    //   unsigned            privateExp : 1;    //14) SET when the private exponent
    //                                          //    of an RSA key has been
validated.
    unsigned not_used_14 : 1;
```

```
    unsigned occupied    : 1;  //15) SET when the slot is occupied.
    unsigned derivation  : 1;  //16) SET when the key is a derivation
                               //        parent
    unsigned external : 1;     //17) SET when the object is loaded with
                               //    TPM2_LoadExternal();
} OBJECT_ATTRIBUTES;

#  if ALG_RSA
// There is an overload of the sensitive.rsa.t.size field of a TPMT_SENSITIVE when an
// RSA key is loaded. When the sensitive->sensitive contains an RSA key with all of
// the CRT values, then the MSB of the size field will be set to indicate that the
// buffer contains all 5 of the CRT private key values.
#    define RSA_prime_flag 0x8000
#  endif

//*** OBJECT Structure
// An OBJECT structure holds the object public, sensitive, and meta-data
// associated. This structure is implementation dependent. For this
// implementation, the structure is not optimized for space but rather
// for clarity of the reference implementation. Other implementations
// may choose to overlap portions of the structure that are not used
// simultaneously. These changes would necessitate changes to the source
// code but those changes would be compatible with the reference
// implementation.

typedef struct OBJECT
{
    // The attributes field is required to be first followed by the publicArea.
    // This allows the overlay of the object structure and a sequence structure
    OBJECT_ATTRIBUTES attributes;     // object attributes
    TPMT_PUBLIC       publicArea;     // public area of an object
    TPMT_SENSITIVE    sensitive;      // sensitive area of an object
    TPM2B_NAME        qualifiedName;  // object qualified name
    TPMI_DH_OBJECT    evictHandle;    // if the object is an evict object,
                                      // the original handle is kept here.
                                      // The 'working' handle will be the
                                      // handle of an object slot.
    TPM2B_NAME name;                  // Name of the object name. Kept here
                                      // to avoid repeatedly computing it.
    TPMI_RH_HIERARCHY hierarchy;      // Hierarchy for the object. While the
                                      // base hierarchy can be deduced from
                                      // 'attributes', if the hierarchy is
                                      // firmware-bound or SVN-bound then
                                      // this field carries additional metadata
                                      // needed to derive the proof value for
                                      // the object.
} OBJECT;

//*** HASH_OBJECT Structure
// This structure holds a hash sequence object or an event sequence object.
//
// The first four components of this structure are manually set to be the same as
// the first four components of the object structure. This prevents the object
// from being inadvertently misused as sequence objects occupy the same memory as
// a regular object. A debug check is present to make sure that the offsets are
// what they are supposed to be.
// NOTE: In a future version, this will probably be renamed as SEQUENCE_OBJECT
typedef struct HASH_OBJECT
{
    OBJECT_ATTRIBUTES attributes;        // The attributes of the HASH object
    TPMI_ALG_PUBLIC   type;              // algorithm
    TPMI_ALG_HASH     nameAlg;           // name algorithm
    TPMA_OBJECT       objectAttributes;  // object attributes

    // The data below is unique to a sequence object
    TPM2B_AUTH auth;  // authorization for use of sequence
```

```
    union
    {
        HASH_STATE hashState[HASH_COUNT];
        HMAC_STATE hmacState;
    } state;
} HASH_OBJECT;

typedef BYTE HASH_OBJECT_BUFFER[sizeof(HASH_OBJECT)];

//*** ANY_OBJECT
// This is the union for holding either a sequence object or a regular object
// for ContextSave and ContextLoad.
typedef union ANY_OBJECT
{
    OBJECT      entity;
    HASH_OBJECT hash;
} ANY_OBJECT;

typedef BYTE ANY_OBJECT_BUFFER[sizeof(ANY_OBJECT)];

//**AUTH_DUP Types
// These values are used in the authorization processing.

typedef UINT32 AUTH_ROLE;
#  define AUTH_NONE  ((AUTH_ROLE)(0))
#  define AUTH_USER  ((AUTH_ROLE)(1))
#  define AUTH_ADMIN ((AUTH_ROLE)(2))
#  define AUTH_DUP   ((AUTH_ROLE)(3))

//** Active Session Context
//*** Description
// The structures in this section define the internal structure of a session
// context.
//
//*** SESSION_ATTRIBUTES
// The attributes in the SESSION_ATTRIBUTES structure track the various properties
// of the session. It maintains most of the tracking state information for the
// policy session. It is used within the SESSION structure.

typedef struct SESSION_ATTRIBUTES
{
    // SET if the session may only be used for policy
    unsigned isPolicy : 1;
    // SET if the session is used for audit
    unsigned isAudit : 1;
    // SET if the session is bound to an entity. This attribute will be CLEAR if
    // either isPolicy or isAudit is SET.
    unsigned isBound : 1;
    // SET if the cpHash has been defined. This attribute is not SET unless
    // 'isPolicy' is SET.
    unsigned isCpHashDefined : 1;
    // SET if the nameHash has been defined. This attribute is not SET unless
    // 'isPolicy' is SET.
    unsigned isNameHashDefined : 1;
    // SET if the pHash has been defined. This attribute is not SET unless
    // 'isPolicy' is SET.
    unsigned isParametersHashDefined : 1;
    // SET if the templateHash needs to be checked for Create, CreatePrimary, or
    // CreateLoaded.
    unsigned isTemplateHashDefined : 1;
    // SET if the authValue is required for computing the session HMAC. This
    // attribute is not SET unless 'isPolicy' is SET.
    unsigned isAuthValueNeeded : 1;
    // SET if a password authValue is required for authorization This attribute
    // is not SET unless 'isPolicy' is SET.
    unsigned isPasswordNeeded : 1;
```

```
    // SET if physical presence is required to be asserted when the
    // authorization is checked. This attribute is not SET unless 'isPolicy' is
    // SET.
    unsigned isPPRequired : 1;
    // SET if the policy session is created for trial of the policy's policyHash
    // generation. This attribute is not SET unless 'isPolicy' is SET.
    unsigned isTrialPolicy : 1;
    // SET if the bind entity had noDA CLEAR. If this is SET, then an
    // authorization failure using this session will count against lockout even
    // if the object being authorized is exempt from DA.
    unsigned isDaBound : 1;
    // SET if the session is bound to lockoutAuth.
    unsigned isLockoutBound : 1;
    // This attribute is SET when the authValue of an object is to be included
    // in the computation of the HMAC key for the command and response
    // computations. (was 'requestWasBound')
    unsigned includeAuth : 1;
    // SET if the TPMA_NV_WRITTEN attribute needs to be checked when the policy
    // is used for authorization for NV access. If this is SET for any other
    // type, the policy will fail.
    unsigned checkNvWritten : 1;
    // SET if TPMA_NV_WRITTEN is required to be SET. Used when 'checkNvWritten'
    // is SET
    unsigned nvWrittenState : 1;
} SESSION_ATTRIBUTES;

//*** IsCpHashUnionOccupied()
// This function indicates whether the session attributes indicate that one of
// the members of the union containing `cpHash` are set.
BOOL IsCpHashUnionOccupied(SESSION_ATTRIBUTES attrs);

//*** SESSION Structure
// The SESSION structure contains all the context of a session except for the
// associated contextID.
//
// Note: The contextID of a session is only relevant when the session context
// is stored off the TPM.

typedef struct SESSION
{
    SESSION_ATTRIBUTES attributes;  // session attributes
    UINT32             pcrCounter;  // PCR counter value when PCR is
                                    // included (policy session)
                                    // If no PCR is included, this
                                    // value is 0.
    UINT64 startTime;               // The value in g_time when the session
                                    // was started (policy session)
    UINT64 timeout;                 // The timeout relative to g_time
                                    // There is no timeout if this value
                                    // is 0.
    CLOCK_NONCE epoch;              // The g_clockEpoch value when the
                                    // session was started. If g_clockEpoch
                                    // does not match this value when the
                                    // timeout is used, then
                                    // then the command will fail.
    TPM_CC        commandCode;      // command code (policy session)
    TPM_ALG_ID    authHashAlg;      // session hash algorithm
    TPMA_LOCALITY commandLocality;  // command locality (policy session)
    TPMT_SYM_DEF  symmetric;        // session symmetric algorithm (if any)
    TPM2B_AUTH    sessionKey;       // session secret value used for
                                    // this session
    TPM2B_NONCE nonceTPM;           // last TPM-generated nonce for
                                    // generating HMAC and encryption keys
    union
    {
        TPM2B_NAME boundEntity;  // value used to track the entity to
```

```
                                    // which the session is bound

        TPM2B_DIGEST cpHash;        // the required cpHash value for the
                                    // command being authorized
        TPM2B_DIGEST nameHash;      // the required nameHash
        TPM2B_DIGEST templateHash;  // the required template for creation
        TPM2B_DIGEST pHash;         // the required parameter hash value for the
                                    // command being authorized
    } u1;

    union
    {
        TPM2B_DIGEST auditDigest;   // audit session digest
        TPM2B_DIGEST policyDigest;  // policyHash
    } u2;                           // audit log and policyHash may
                                    // share space to save memory
} SESSION;

#  define EXPIRES_ON_RESET   INT32_MIN
#  define TIMEOUT_ON_RESET   UINT64_MAX
#  define EXPIRES_ON_RESTART (INT32_MIN + 1)
#  define TIMEOUT_ON_RESTART (UINT64_MAX - 1)

typedef BYTE SESSION_BUF[sizeof(SESSION)];

//**********************************************************************************
//** PCR
//**********************************************************************************
//***PCR_SAVE Structure
// The PCR_SAVE structure type contains the PCR data that are saved across power
// cycles. Only the static PCR are required to be saved across power cycles. The
// DRTM and resettable PCR are not saved. The number of static and resettable PCR
// is determined by the platform-specific specification to which the TPM is built.

#  define PCR_SAVE_SPACE(HASH, Hash) BYTE Hash[NUM_STATIC_PCR][HASH##_DIGEST_SIZE];

typedef struct PCR_SAVE
{
    FOR_EACH_HASH(PCR_SAVE_SPACE)

    // This counter increments whenever the PCR are updated.
    // NOTE: A platform-specific specification may designate
    //       certain PCR changes as not causing this counter
    //       to increment.
    UINT32 pcrCounter;
} PCR_SAVE;

//***PCR_POLICY
#  if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
// This structure holds the PCR policies, one for each group of PCR controlled
// by policy.
typedef struct PCR_POLICY
{
    TPMI_ALG_HASH hashAlg[NUM_POLICY_PCR_GROUP];
    TPM2B_DIGEST  a;
    TPM2B_DIGEST  policy[NUM_POLICY_PCR_GROUP];
} PCR_POLICY;
#  endif

//***PCR_AUTHVALUE
// This structure holds the PCR policies, one for each group of PCR controlled
// by policy.
typedef struct PCR_AUTH_VALUE
{
    TPM2B_DIGEST auth[NUM_AUTHVALUE_PCR_GROUP];
} PCR_AUTHVALUE;
```

```
//**STARTUP_TYPE
// This enumeration is the possible startup types. The type is determined
// by the combination of TPM2_ShutDown and TPM2_Startup.
typedef enum
{
    SU_RESET,
    SU_RESTART,
    SU_RESUME
} STARTUP_TYPE;

//**NV

//***NV_INDEX
// The NV_INDEX structure defines the internal format for an NV index.
// The 'indexData' size varies according to the type of the index.
// In this implementation, all of the index is manipulated as a unit.
// NOTE: In this implementation of the TPM, the extended bits are always 0.
// Therefore, they are stored in the NV subsystem as legacy structures,
// even when the handle type indicates that the index can have extended
// attributes.
typedef struct NV_INDEX
{
    TPMS_NV_PUBLIC publicArea;
    TPM2B_AUTH     authValue;
} NV_INDEX;

//*** NV_REF
// An NV_REF is an opaque value returned by the NV subsystem. It is used to
// reference and NV Index in a relatively efficient way. Rather than having to
// continually search for an Index, its reference value may be used. In this
// implementation, an NV_REF is a byte pointer that points to the copy of the
// NV memory that is kept in RAM.
typedef UINT32 NV_REF;

typedef BYTE*  NV_RAM_REF;
//***NV_PIN
// This structure deals with the possible endianess differences between the
// canonical form of the TPMS_NV_PIN_COUNTER_PARAMETERS structure and the internal
// value. The structures allow the data in a PIN index to be read as an 8-octet
// value using NvReadUINT64Data(). That function will byte swap all the values on a
// little endian system. This will put the bytes with the 4-octet values in the
// correct order but will swap the pinLimit and pinCount values. When written, the
// PIN index is simply handled as a normal index with the octets in canonical order.
#  if BIG_ENDIAN_TPM
typedef struct
{
    UINT32 pinCount;
    UINT32 pinLimit;
} PIN_DATA;
#  else
typedef struct
{
    UINT32 pinLimit;
    UINT32 pinCount;
} PIN_DATA;
#  endif

typedef union
{
    UINT64   intVal;
    PIN_DATA pin;
} NV_PIN;

//**COMMIT_INDEX_MASK
// This is the define for the mask value that is used when manipulating
```

```
// the bits in the commit bit array. The commit counter is a 64-bit
// value and the low order bits are used to index the commitArray.
// This mask value is applied to the commit counter to extract the
// bit number in the array.
#  if ALG_ECC

#    define COMMIT_INDEX_MASK ((UINT16)((sizeof(gr.commitArray) * 8) - 1))

#  endif

//**********************************************************************
//**********************************************************************
//** RAM Global Values
//**********************************************************************
//**********************************************************************
//*** Description
// The values in this section are only extant in RAM or ROM as constant values.

//*** Crypto Self-Test Values
EXTERN ALGORITHM_VECTOR g_implementedAlgorithms;
EXTERN ALGORITHM_VECTOR g_toTest;

//*** g_rcIndex[]
// This array is used to contain the array of values that are added to a return
// code when it is a parameter-, handle-, or session-related error.
// This is an implementation choice and the same result can be achieved by using
// a macro.
extern const UINT16 g_rcIndex[15];

//*** g_exclusiveAuditSession
// This location holds the session handle for the current exclusive audit
// session. If there is no exclusive audit session, the location is set to
// TPM_RH_UNASSIGNED.
EXTERN TPM_HANDLE g_exclusiveAuditSession;

//*** g_time
// This is the value in which we keep the current command time. This is initialized
// at the start of each command. The time is the accumulated time since the last
// time that the TPM's timer was last powered up. Clock is the accumulated time
// since the last time that the TPM was cleared. g_time is in mS.
EXTERN UINT64 g_time;

//*** g_timeEpoch
// This value contains the current clock Epoch. It changes when there is a clock
// discontinuity. It may be necessary to place this in NV should the timer be able
// to run across a power down of the TPM but not in all cases (e.g. dead battery).
// If the nonce is placed in NV, it should go in gp because it should be changing
// slowly.
#  if CLOCK_STOPS
EXTERN CLOCK_NONCE g_timeEpoch;
#  else
#    define g_timeEpoch gp.timeEpoch
#  endif

//*** g_phEnable
// This is the platform hierarchy control and determines if the platform hierarchy
// is available. This value is SET on each TPM2_Startup(). The default value is
// SET.
EXTERN BOOL g_phEnable;

//*** g_pcrReConfig
// This value is SET if a TPM2_PCR_Allocate command successfully executed since
// the last TPM2_Startup(). If so, then the next shutdown is required to be
// Shutdown(CLEAR).
EXTERN BOOL g_pcrReConfig;
```

```
//*** g_DRTMHandle
// This location indicates the sequence object handle that holds the DRTM
// sequence data. When not used, it is set to TPM_RH_UNASSIGNED. A sequence
// DRTM sequence is started on either _TPM_Init or _TPM_Hash_Start.
EXTERN TPMI_DH_OBJECT g_DRTMHandle;

//*** g_DrtmPreStartup
// This value indicates that an H-CRTM occurred after _TPM_Init but before
// TPM2_Startup(). The define for PRE_STARTUP_FLAG is used to add the
// g_DrtmPreStartup value to gp_orderlyState at shutdown. This hack is to avoid
// adding another NV variable.
EXTERN BOOL g_DrtmPreStartup;

//*** g_StartupLocality3
// This value indicates that a TPM2_Startup() occurred at locality 3. Otherwise, it
// at locality 0. The define for STARTUP_LOCALITY_3 is to
// indicate that the startup was not at locality 0. This hack is to avoid
// adding another NV variable.
EXTERN BOOL g_StartupLocality3;

//***TPM_SU_NONE
// Part 2 defines the two shutdown/startup types that may be used in
// TPM2_Shutdown() and TPM2_Starup(). This additional define is
// used by the TPM to indicate that no shutdown was received.
// NOTE: This is a reserved value.
#  define SU_NONE_VALUE (0xFFFF)
#  define TPM_SU_NONE   (TPM_SU)(SU_NONE_VALUE)

//*** TPM_SU_DA_USED
// As with TPM_SU_NONE, this value is added to allow indication that the shutdown
// was not orderly and that a DA=protected object was reference during the previous
// cycle.
#  define SU_DA_USED_VALUE (SU_NONE_VALUE - 1)
#  define TPM_SU_DA_USED   (TPM_SU)(SU_DA_USED_VALUE)

//*** Startup Flags
// These flags are included in gp.orderlyState. These are hacks and are being
// used to avoid having to change the layout of gp. The PRE_STARTUP_FLAG indicates
// that a _TPM_Hash_Start/_Data/_End sequence was received after _TPM_Init but
// before TPM2_StartUp(). STARTUP_LOCALITY_3 indicates that the last TPM2_Startup()
// was received at locality 3. These flags are only  relevant if after a
// TPM2_Shutdown(STATE).
#  define PRE_STARTUP_FLAG   0x8000
#  define STARTUP_LOCALITY_3 0x4000

#  if USE_DA_USED
//*** g_daUsed
// This location indicates if a DA-protected value is accessed during a boot
// cycle. If none has, then there is no need to increment 'failedTries' on the
// next non-orderly startup. This bit is merged with gp.orderlyState when
// gp.orderly is set to SU_NONE_VALUE
EXTERN BOOL g_daUsed;
#  endif

//*** g_updateNV
// This flag indicates if NV should be updated at the end of a command.
// This flag is set to UT_NONE at the beginning of each command in ExecuteCommand().
// This flag is checked in ExecuteCommand() after the detailed actions of a command
// complete. If the command execution was successful and this flag is not UT_NONE,
// any pending NV writes will be committed to NV.
// UT_ORDERLY causes any RAM data to be written to the orderly space for staging
// the write to NV.
typedef BYTE UPDATE_TYPE;
#  define UT_NONE    (UPDATE_TYPE)0
#  define UT_NV      (UPDATE_TYPE)1
#  define UT_ORDERLY (UPDATE_TYPE)(UT_NV + 2)
```

```
EXTERN UPDATE_TYPE g_updateNV;

//*** g_powerWasLost
// This flag is used to indicate if the power was lost. It is SET in _TPM__Init.
// This flag is cleared by TPM2_Startup() after all power-lost activities are
// completed.
// Note: When power is applied, this value can come up as anything. However,
// _plat__WasPowerLost() will provide the proper indication in that case. So, when
// power is actually lost, we get the correct answer. When power was not lost, but
// the power-lost processing has not been completed before the next _TPM_Init(),
// then the TPM still does the correct thing.
EXTERN BOOL g_powerWasLost;

//*** g_clearOrderly
// This flag indicates if the execution of a command should cause the orderly
// state to be cleared.  This flag is set to FALSE at the beginning of each
// command in ExecuteCommand() and is checked in ExecuteCommand() after the
// detailed actions of a command complete but before the check of
// 'g_updateNV'. If this flag is TRUE, and the orderly state is not
// SU_NONE_VALUE, then the orderly state in NV memory will be changed to
// SU_NONE_VALUE or SU_DA_USED_VALUE.
EXTERN BOOL g_clearOrderly;

//*** g_prevOrderlyState
// This location indicates how the TPM was shut down before the most recent
// TPM2_Startup(). This value, along with the startup type, determines if
// the TPM should do a TPM Reset, TPM Restart, or TPM Resume.
EXTERN TPM_SU g_prevOrderlyState;

//*** g_nvOk
// This value indicates if the NV integrity check was successful or not. If not and
// the failure was severe, then the TPM would have been put into failure mode after
// it had been re-manufactured. If the NV failure was in the area where the state-save
// data is kept, then this variable will have a value of FALSE indicating that
// a TPM2_Startup(CLEAR) is required.
EXTERN BOOL g_nvOk;
// NV availability is sampled as the start of each command and stored here
// so that its value remains consistent during the command execution
EXTERN TPM_RC g_NvStatus;

//*** g_platformUnique

// This location contains unique value(s) used by the TPM Platform vendor.
// These are loaded on every _TPM2_Startup() using the _plat__GetUnique function.
// The "which" parameter to  _plat__GetUnique indicates the value to return.
// If used, the TPM vendor is expected to use these values for authentication.
#  if VENDOR_PERMANENT_AUTH_ENABLED == YES
// which = 1, the authorization value for VENDOR_PERMANENT_AUTH_HANDLE
EXTERN TPM2B_AUTH g_platformUniqueAuth;
#  endif

//**********************************************************************************
//**********************************************************************************
//** Persistent Global Values
//**********************************************************************************
//**********************************************************************************
//*** Description
// The values in this section are global values that are persistent across power
// events. The lifetime of the values determines the structure in which the value
// is placed.

//**********************************************************************************
//*** PERSISTENT_DATA
//**********************************************************************************
// This structure holds the persistent values that only change as a consequence
// of a specific Protected Capability and are not affected by TPM power events
```

```
// (TPM2_Startup() or TPM2_Shutdown().
typedef struct
{
    // data provided by the platform library during manufacturing.
    // Opaque to the TPM Core library, but may be used by the platform library.
    BYTE platformReserved[PERSISTENT_DATA_PLATFORM_SPACE];


//*****************************************************************************
    //          Hierarchy

//*****************************************************************************
    // The values in this section are related to the hierarchies.

    BOOL disableClear;  // TRUE if TPM2_Clear() using
                        // lockoutAuth is disabled

    // Hierarchy authPolicies
    TPMI_ALG_HASH ownerAlg;
    TPMI_ALG_HASH endorsementAlg;
    TPMI_ALG_HASH lockoutAlg;
    TPM2B_DIGEST  ownerPolicy;
    TPM2B_DIGEST  endorsementPolicy;
    TPM2B_DIGEST  lockoutPolicy;

    // Hierarchy authValues
    TPM2B_AUTH ownerAuth;
    TPM2B_AUTH endorsementAuth;
    TPM2B_AUTH lockoutAuth;

    // Primary Seeds
    TPM2B_SEED EPSeed;
    TPM2B_SEED SPSeed;
    TPM2B_SEED PPSeed;
    // Note there is a nullSeed in the state_reset memory.

    // Hierarchy proofs
    TPM2B_PROOF phProof;
    TPM2B_PROOF shProof;
    TPM2B_PROOF ehProof;
    // Note there is a nullProof in the state_reset memory.


//*****************************************************************************
    //          Reset Events

//*****************************************************************************
    // A count that increments at each TPM reset and never get reset during the life
    // time of TPM.  The value of this counter is initialized to 1 during TPM
    // manufacture process. It is used to invalidate all saved contexts after a TPM
    // Reset.
    UINT64 totalResetCount;

    // This counter increments on each TPM Reset. The counter is reset by
    // TPM2_Clear().
    UINT32 resetCount;

//*****************************************************************************
//          PCR
//*****************************************************************************
// This structure hold the policies for those PCR that have an update policy.
// This implementation only supports a single group of PCR controlled by
// policy. If more are required, then this structure would be changed to
// an array.
#  if defined  NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
    PCR_POLICY pcrPolicies;
```

```
#  endif

    // This structure indicates the allocation of PCR. The structure contains a
    // list of PCR allocations for each implemented algorithm. If no PCR are
    // allocated for an algorithm, a list entry still exists but the bit map
    // will contain no SET bits.
    TPML_PCR_SELECTION pcrAllocated;


//**********************************************************************************
    //          Physical Presence

//**********************************************************************************
    // The PP_LIST type contains a bit map of the commands that require physical
    // to be asserted when the authorization is evaluated. Physical presence will be
    // checked if the corresponding bit in the array is SET and if the authorization
    // handle is TPM_RH_PLATFORM.
    //
    // These bits may be changed with TPM2_PP_Commands().
    BYTE ppList[(COMMAND_COUNT + 7) / 8];


//**********************************************************************************
    //          Dictionary attack values

//**********************************************************************************
    // These values are used for dictionary attack tracking and control.
    UINT32 failedTries;  // the current count of unexpired
                         // authorization failures

    UINT32 maxTries;  // number of unexpired authorization
                      // failures before the TPM is in
                      // lockout

    UINT32 recoveryTime;  // time between authorization failures
                          // before failedTries is decremented

    UINT32 lockoutRecovery;  // time that must expire between
                             // authorization failures associated
                             // with lockoutAuth

    BOOL lockOutAuthEnabled;  // TRUE if use of lockoutAuth is
                              // allowed

    //*****************************************************************************
    //            Orderly State
    //*****************************************************************************
    // The orderly state for current cycle
    TPM_SU orderlyState;

    //*****************************************************************************
    //           Command audit values.
    //*****************************************************************************
    BYTE          auditCommands[((COMMAND_COUNT + 1) + 7) / 8];
    TPMI_ALG_HASH auditHashAlg;
    UINT64        auditCounter;

    //*****************************************************************************
    //           Algorithm selection
    //*****************************************************************************
    //
    // The 'algorithmSet' value indicates the collection of algorithms that are
    // currently in used on the TPM.  The interpretation of value is vendor dependent.
    UINT32 algorithmSet;

    //*****************************************************************************
```

```
    //           Firmware version
    //*****************************************************************************
    // The firmwareV1 and firmwareV2 values are instanced in TimeStamp.c. This is
    // a scheme used in development to allow determination of the linker build time
    // of the TPM. An actual implementation would implement these values in a way that
    // is consistent with vendor needs. The values are maintained in RAM for
simplified
    // access with a master version in NV.  These values are modified in a
    // vendor-specific way.

    // g_firmwareV1 contains the more significant 32-bits of the vendor version
number.
    // In the reference implementation, if this value is printed as a hex
    // value, it will have the format of YYYYMMDD
    UINT32 firmwareV1;

    // g_firmwareV1 contains the less significant 32-bits of the vendor version
number.
    // In the reference implementation, if this value is printed as a hex
    // value, it will have the format of 00 HH MM SS
    UINT32 firmwareV2;
//*****************************************************************************
//           Timer Epoch
//*****************************************************************************
// timeEpoch contains a nonce that has a vendor=specific size (should not be
// less than 8 bytes. This nonce changes when the clock epoch changes. The clock
// epoch changes when there is a discontinuity in the timing of the TPM.
#  if !CLOCK_STOPS
    CLOCK_NONCE timeEpoch;
#  endif

} PERSISTENT_DATA;

EXTERN PERSISTENT_DATA gp;

//*****************************************************************************
//*****************************************************************************
//*** ORDERLY_DATA
//*****************************************************************************
//*****************************************************************************
// The data in this structure is saved to NV on each TPM2_Shutdown().
typedef struct orderly_data
{
    //*****************************************************************************
    //           TIME
    //*****************************************************************************

    // Clock has two parts. One is the state save part and one is the NV part. The
    // state save version is updated on each command. When the clock rolls over, the
    // NV version is updated. When the TPM starts up, if the TPM was shutdown in and
    // orderly way, then the sClock value is used to initialize the clock. If the
    // TPM shutdown was not orderly, then the persistent value is used and the safe
    // attribute is clear.

    UINT64      clock;      // The orderly version of clock
    TPMI_YES_NO clockSafe;  // Indicates if the clock value is
                            // safe.

    // In many implementations, the quality of the entropy available is not that
    // high. To compensate, the current value of the drbgState can be saved and
    // restored on each power cycle. This prevents the internal state from reverting
    // to the initial state on each power cycle and starting with a limited amount
    // of entropy. By keeping the old state and adding entropy, the entropy will
    // accumulate.
    DRBG_STATE drbgState;
```

```
// These values allow the accumulation of self-healing time across orderly shutdown
// of the TPM.
#  if ACCUMULATE_SELF_HEAL_TIMER
    UINT64 selfHealTimer;  // current value of s_selfHealTimer
    UINT64 lockoutTimer;   // current value of s_lockoutTimer
    UINT64 time;           // current value of g_time at shutdown
#  endif                  // ACCUMULATE_SELF_HEAL_TIMER

// These are the ACT Timeout values. They are saved with the other timers
#  define DefineActData(N) ACT_STATE ACT_##N;
    FOR_EACH_ACT(DefineActData)

    // this is the 'signaled' attribute data for all the ACT. It is done this way so
    // that they can be manipulated by ACT number rather than having to access a
    // structure.
    UINT16 signaledACT;
    UINT16 preservedSignaled;

#  if ORDERLY_DATA_PADDING != 0
    BYTE reserved[ORDERLY_DATA_PADDING];
#  endif

} ORDERLY_DATA;

#  if ACCUMULATE_SELF_HEAL_TIMER
#    define s_selfHealTimer go.selfHealTimer
#    define s_lockoutTimer  go.lockoutTimer
#  endif  // ACCUMULATE_SELF_HEAL_TIMER

#  define drbgDefault go.drbgState

EXTERN ORDERLY_DATA go;

//*****************************************************************************
//*****************************************************************************
//*** STATE_CLEAR_DATA
//*****************************************************************************
//*****************************************************************************
// This structure contains the data that is saved on Shutdown(STATE)
// and restored on Startup(STATE).  The values are set to their default
// settings on any Startup(Clear). In other words, the data is only persistent
// across TPM Resume.
//
// If the comments associated with a parameter indicate a default reset value, the
// value is applied on each Startup(CLEAR).

typedef struct state_clear_data
{
    //*************************************************************************
    //           Hierarchy Control
    //*************************************************************************
    BOOL          shEnable;        // default reset is SET
    BOOL          ehEnable;        // default reset is SET
    BOOL          phEnableNV;      // default reset is SET
    TPMI_ALG_HASH platformAlg;     // default reset is TPM_ALG_NULL
    TPM2B_DIGEST  platformPolicy;  // default reset is an Empty Buffer
    TPM2B_AUTH    platformAuth;    // default reset is an Empty Buffer

    //*************************************************************************
    //           PCR
    //*************************************************************************
    // The set of PCR to be saved on Shutdown(STATE)
    PCR_SAVE pcrSave;  // default reset is 0...0

    // This structure hold the authorization values for those PCR that have an
    // update authorization.
```

```
    // This implementation only supports a single group of PCR controlled by
    // authorization. If more are required, then this structure would be changed to
    // an array.
    PCR_AUTHVALUE pcrAuthValues;

//*****************************************************************************
//           ACT
//*****************************************************************************
#  define DefineActPolicySpace(N) TPMT_HA act_##N;
    FOR_EACH_ACT(DefineActPolicySpace)

#  if STATE_CLEAR_DATA_PADDING != 0
    BYTE reserved[STATE_CLEAR_DATA_PADDING];
#  endif
} STATE_CLEAR_DATA;

EXTERN STATE_CLEAR_DATA gc;

//*****************************************************************************
//*****************************************************************************
//***  State Reset Data
//*****************************************************************************
//*****************************************************************************
// This structure contains data is that is saved on Shutdown(STATE) and restored on
// the subsequent Startup(ANY). That is, the data is preserved across TPM Resume
// and TPM Restart.
//
// If a default value is specified in the comments this value is applied on
// TPM Reset.

typedef struct state_reset_data
{
    //*************************************************************************
    //          Hierarchy Control
    //*************************************************************************
    TPM2B_PROOF nullProof;  // The proof value associated with
                            // the TPM_RH_NULL hierarchy. The
                            // default reset value is from the RNG.

    TPM2B_SEED nullSeed;  // The seed value for the TPM_RN_NULL
                          // hierarchy. The default reset value
                          // is from the RNG.

    //*************************************************************************
    //           Context
    //*************************************************************************
    // The 'clearCount' counter is incremented each time the TPM successfully executes
    // a TPM Resume. The counter is included in each saved context that has 'stClear'
    // SET (including descendants of keys that have 'stClear' SET). This prevents
these
    // objects from being loaded after a TPM Resume.
    // If 'clearCount' is at its maximum value when the TPM receives a
Shutdown(STATE),
    // the TPM will return TPM_RC_RANGE and the TPM will only accept Shutdown(CLEAR).
    UINT32 clearCount;  // The default reset value is 0.

    UINT64 objectContextID;  // This is the context ID for a saved
                             //  object context. The default reset
                             //  value is 0.
    CONTEXT_SLOT contextArray[MAX_ACTIVE_SESSIONS];  // This array contains
        // contains the values used to track
        // the version numbers of saved
        // contexts (see
        // Session.c in for details). The
        // default reset value is {0}.
```

```
    CONTEXT_COUNTER contextCounter;  // This is the value from which the
                                     // 'contextID' is derived. The
                                     // default reset value is {0}.

    //*****************************************************************************
    //           Command Audit
    //*****************************************************************************
    // When an audited command completes, ExecuteCommand() checks the return
    // value.  If it is TPM_RC_SUCCESS, and the command is an audited command, the
    // TPM will extend the cpHash and rpHash for the command to this value. If this
    // digest was the Zero Digest before the cpHash was extended, the audit counter
    // is incremented.

    TPM2B_DIGEST commandAuditDigest;  // This value is set to an Empty Digest
                                      // by TPM2_GetCommandAuditDigest() or a
                                      // TPM Reset.

    //*****************************************************************************
    //           Boot counter
    //*****************************************************************************

    UINT32 restartCount;  // This counter counts TPM Restarts.
                          // The default reset value is 0.


//*********************************************************************************
    //            PCR

//*********************************************************************************
    // This counter increments whenever the PCR are updated. This counter is preserved
    // across TPM Resume even though the PCR are not preserved. This is because
    // sessions remain active across TPM Restart and the count value in the session
    // is compared to this counter so this counter must have values that are unique
    // as long as the sessions are active.
    // NOTE: A platform-specific specification may designate that certain PCR changes
    //       do not increment this counter to increment.
    UINT32 pcrCounter;  // The default reset value is 0.

#  if ALG_ECC

    //*****************************************************************************
    //         ECDAA
    //*****************************************************************************
    UINT64 commitCounter;  // This counter increments each time
                           // TPM2_Commit() returns
                           // TPM_RC_SUCCESS. The default reset
                           // value is 0.

    TPM2B_NONCE commitNonce;  // This random value is used to compute
                              // the commit values. The default reset
                              // value is from the RNG.

    // This implementation relies on the number of bits in g_commitArray being a
    // power of 2 (8, 16, 32, 64, etc.) and no greater than 64K.
    BYTE commitArray[16];  // The default reset value is {0}.

#  endif  // ALG_ECC
#  if STATE_RESET_DATA_PADDING != 0
    BYTE reserved[STATE_RESET_DATA_PADDING];
#  endif
} STATE_RESET_DATA;

EXTERN STATE_RESET_DATA gr;

//** NV Layout
// The NV data organization is
```

```
// 1) a PERSISTENT_DATA structure
// 2) a STATE_RESET_DATA structure
// 3) a STATE_CLEAR_DATA structure
// 4) an ORDERLY_DATA structure
// 5) the user defined NV index space
#  define NV_PERSISTENT_DATA  (0)
#  define NV_STATE_RESET_DATA (NV_PERSISTENT_DATA + sizeof(PERSISTENT_DATA))
#  define NV_STATE_CLEAR_DATA (NV_STATE_RESET_DATA + sizeof(STATE_RESET_DATA))
#  define NV_ORDERLY_DATA     (NV_STATE_CLEAR_DATA + sizeof(STATE_CLEAR_DATA))
#  define NV_INDEX_RAM_DATA   (NV_ORDERLY_DATA + sizeof(ORDERLY_DATA))
#  define NV_USER_DYNAMIC     (NV_INDEX_RAM_DATA + sizeof(s_indexOrderlyRam))
#  define NV_USER_DYNAMIC_END NV_MEMORY_SIZE

//** Global Macro Definitions
// The NV_READ_PERSISTENT and NV_WRITE_PERSISTENT macros are used to access members
// of the PERSISTENT_DATA structure in NV.
#  define NV_READ_PERSISTENT(to, from) \
      NvRead(&to, offsetof(PERSISTENT_DATA, from), sizeof(to))

#  define NV_WRITE_PERSISTENT(to, from) \
      NvWrite(offsetof(PERSISTENT_DATA, to), sizeof(gp.to), &from)

#  define CLEAR_PERSISTENT(item) \
      NvClearPersistent(offsetof(PERSISTENT_DATA, item), sizeof(gp.item))

#  define NV_SYNC_PERSISTENT(item) NV_WRITE_PERSISTENT(item, gp.item)

// At the start of command processing, the index of the command is determined. This
// index value is used to access the various data tables that contain per-command
// information. There are multiple options for how the per-command tables can be
// implemented. This is resolved in GetClosestCommandIndex().
typedef UINT16 COMMAND_INDEX;
#  define UNIMPLEMENTED_COMMAND_INDEX ((COMMAND_INDEX)(~0))

typedef struct _COMMAND_FLAGS_
{
    unsigned trialPolicy : 1;  //1) If SET, one of the handles references a
                               //   trial policy and authorization may be
                               //   skipped. This is only allowed for a policy
                               //   command.
} COMMAND_FLAGS;

// This structure is used to avoid having to manage a large number of
// parameters being passed through various levels of the command input processing.
//

// The following macros are used to define the space for the CP and RP hashes. Space,
// is provided for each implemented hash algorithm because it is not known what the
// caller may use.
#  define CP_HASH(HASH, Hash) TPM2B_##HASH##_DIGEST Hash##CpHash;
#  define RP_HASH(HASH, Hash) TPM2B_##HASH##_DIGEST Hash##RpHash;

typedef struct COMMAND
{
    TPM_ST        tag;                    // the parsed command tag
    TPM_CC        code;                   // the parsed command code
    COMMAND_INDEX index;                  // the computed command index
    UINT32        handleNum;              // the number of entity handles in the
                                          //   handle area of the command
    TPM_HANDLE handles[MAX_HANDLE_NUM];   // the parsed handle values
    UINT32     sessionNum;                // the number of sessions found
    INT32      parameterSize;             // starts out with the parsed command size
                                          // and is reduced and values are
                                          // unmarshaled. Just before calling the
                                          // command actions, this should be zero.
                                          // After the command actions, this number
```

```
                                        // should grow as values are marshaled
                                        // in to the response buffer.
    INT32 authSize;                     // this is initialized with the parsed size
                                        // of authorizationSize field and should
                                        // be zero when the authorizations are
                                        // parsed.
    BYTE* parameterBuffer;              // input to ExecuteCommand
    BYTE* responseBuffer;               // input to ExecuteCommand
    FOR_EACH_HASH(CP_HASH)              // space for the CP hashes
    FOR_EACH_HASH(RP_HASH)              // space for the RP hashes
} COMMAND;

// TPM2B String constants used for KDFs.
// actual definition in global.c
extern const TPM2B* PRIMARY_OBJECT_CREATION;
extern const TPM2B* CFB_KEY;
extern const TPM2B* CONTEXT_KEY;
extern const TPM2B* INTEGRITY_KEY;
extern const TPM2B* SECRET_KEY;
extern const TPM2B* HIERARCHY_PROOF_SECRET_LABEL;
extern const TPM2B* HIERARCHY_SEED_SECRET_LABEL;
extern const TPM2B* HIERARCHY_FW_SECRET_LABEL;
extern const TPM2B* HIERARCHY_SVN_SECRET_LABEL;
extern const TPM2B* SESSION_KEY;
extern const TPM2B* STORAGE_KEY;
extern const TPM2B* XOR_KEY;
extern const TPM2B* COMMIT_STRING;
extern const TPM2B* DUPLICATE_STRING;
extern const TPM2B* IDENTITY_STRING;
extern const TPM2B* OBFUSCATE_STRING;
#  if ENABLE_SELF_TESTS
extern const TPM2B* OAEP_TEST_STRING;
#  endif  // ENABLE_SELF_TESTS

//**********************************************************************
//** From CryptTest.c
//**********************************************************************
// This structure contains the self-test state values for the cryptographic modules.
EXTERN CRYPTO_SELF_TEST_STATE g_cryptoSelfTestState;

//**********************************************************************
//** From Manufacture.c
//**********************************************************************
extern BOOL g_manufactured;

// This value indicates if a TPM2_Startup commands has been
// receive since the power on event.  This flag is maintained in power
// simulation module because this is the only place that may reliably set this
// flag to FALSE.
EXTERN BOOL g_initialized;

//** Private data

//**********************************************************************
//*** From SessionProcess.c
//**********************************************************************
#  if defined SESSION_PROCESS_C || defined GLOBAL_C || defined MANUFACTURE_C
// The following arrays are used to save command sessions information so that the
// command handle/session buffer does not have to be preserved for the duration of
// the command. These arrays are indexed by the session index in accordance with
// the order of sessions in the session area of the command.
//
// Array of the authorization session handles
EXTERN TPM_HANDLE s_sessionHandles[MAX_SESSION_NUM];

// Array of authorization session attributes
```

```
EXTERN TPMA_SESSION s_attributes[MAX_SESSION_NUM];

// Array of handles authorized by the corresponding authorization sessions;
// and if none, then TPM_RH_UNASSIGNED value is used
EXTERN TPM_HANDLE s_associatedHandles[MAX_SESSION_NUM];

// Array of nonces provided by the caller for the corresponding sessions
EXTERN TPM2B_NONCE s_nonceCaller[MAX_SESSION_NUM];

// Array of authorization values (HMAC's or passwords) for the corresponding
// sessions
EXTERN TPM2B_AUTH s_inputAuthValues[MAX_SESSION_NUM];

// Array of pointers to the SESSION structures for the sessions in a command
EXTERN SESSION* s_usedSessions[MAX_SESSION_NUM];

// Special value to indicate an undefined session index
#    define UNDEFINED_INDEX (0xFFFF)

// Index of the session used for encryption of a response parameter
EXTERN UINT32 s_encryptSessionIndex;

// Index of the session used for decryption of a command parameter
EXTERN UINT32 s_decryptSessionIndex;

// Index of a session used for audit
EXTERN UINT32 s_auditSessionIndex;

// The cpHash for command audit
#    if CC_GetCommandAuditDigest
EXTERN TPM2B_DIGEST s_cpHashForCommandAudit;
#    endif

// Flag indicating if NV update is pending for the lockOutAuthEnabled or
// failedTries DA parameter
EXTERN BOOL s_DAPendingOnNV;

#  endif  // SESSION_PROCESS_C

//**********************************************************************************
//*** From DA.c
//**********************************************************************************
#  if defined DA_C || defined GLOBAL_C || defined MANUFACTURE_C
// This variable holds the accumulated time since the last time
// that 'failedTries' was decremented. This value is in millisecond.
#    if !ACCUMULATE_SELF_HEAL_TIMER
EXTERN UINT64 s_selfHealTimer;

// This variable holds the accumulated time that the lockoutAuth has been
// blocked.
EXTERN UINT64 s_lockoutTimer;
#    endif  // ACCUMULATE_SELF_HEAL_TIMER

#  endif  // DA_C

//**********************************************************************************
//*** From NV.c
//**********************************************************************************
#  if defined NV_C || defined GLOBAL_C
// This marks the end of the NV area. This is a run-time variable as it might
// not be compile-time constant.
EXTERN NV_REF s_evictNvEnd;

// This space is used to hold the index data for an orderly Index. It also contains
// the attributes for the index.
EXTERN BYTE s_indexOrderlyRam[RAM_INDEX_SPACE];  // The orderly NV Index data
```

```
// This value contains the current max counter value. It is written to the end of
// allocatable NV space each time an index is deleted or added. This value is
// initialized on Startup. The indices are searched and the maximum of all the
// current counter indices and this value is the initial value for this.
EXTERN UINT64 s_maxCounter;

// This is space used for the NV Index cache. As with a persistent object, the
// contents of a referenced index are copied into the cache so that the
// NV Index memory scanning and data copying can be reduced.
// Only code that operates on NV Index data should use this cache directly. When
// that action code runs, s_lastNvIndex will contain the index header information.
// It will have been loaded when the handles were verified.
// NOTE: An NV index handle can appear in many commands that do not operate on the
// NV data (e.g. TPM2_StartAuthSession). However, only one NV Index at a time is
// ever directly referenced by any command. If that changes, then the NV Index
// caching needs to be changed to accommodate that. Currently, the code will verify
// that only one NV Index is referenced by the handles of the command.
EXTERN NV_INDEX s_cachedNvIndex;
EXTERN NV_REF   s_cachedNvRef;
EXTERN BYTE*    s_cachedNvRamRef;

// Initial NV Index/evict object iterator value
#    define NV_REF_INIT (NV_REF)0xFFFFFFFF

#  endif

//**********************************************************************
//*** From Object.c
//**********************************************************************
#  if defined OBJECT_C || defined GLOBAL_C
// This type is the container for an object.

EXTERN OBJECT s_objects[MAX_LOADED_OBJECTS];

#  endif  // OBJECT_C

//**********************************************************************
//*** From PCR.c
//**********************************************************************
#  if defined PCR_C || defined GLOBAL_C
#    include <platform_interface/pcrstruct.h>

EXTERN PCR s_pcrs[IMPLEMENTATION_PCR];

#  endif  // PCR_C

//**********************************************************************
//*** From Session.c
//**********************************************************************
#  if defined SESSION_C || defined GLOBAL_C
// Container for HMAC or policy session tracking information
typedef struct
{
    BOOL    occupied;
    SESSION session;  // session structure
} SESSION_SLOT;

EXTERN SESSION_SLOT s_sessions[MAX_LOADED_SESSIONS];

//  The index in contextArray that has the value of the oldest saved session
//  context. When no context is saved, this will have a value that is greater
//  than or equal to MAX_ACTIVE_SESSIONS.
EXTERN UINT32 s_oldestSavedSession;

// The number of available session slot openings.  When this is 1,
```

```
// a session can't be created or loaded if the GAP is maxed out.
// The exception is that the oldest saved session context can always
// be loaded (assuming that there is a space in memory to put it)
EXTERN int s_freeSessionSlots;

#  endif  // SESSION_C

//**********************************************************************
//*** From IoBuffers.c
//**********************************************************************
#  if defined IO_BUFFER_C || defined GLOBAL_C
// Each command function is allowed a structure for the inputs to the function and
// a structure for the outputs. The command dispatch code unmarshals the input butter
// to the command action input structure starting at the first byte of
// s_actionIoBuffer. The value of s_actionIoAllocation is the number of UINT64 values
// allocated. It is used to set the pointer for the response structure. The command
// dispatch code will marshal the response values into the final output buffer.
EXTERN UINT64 s_actionIoBuffer[768];  // action I/O buffer
EXTERN UINT32 s_actionIoAllocation;   // number of UIN64 allocated for the
                                      // action input structure
#  endif                             // IO_BUFFER_C

//**********************************************************************
//*** From TPMFail.c
//**********************************************************************
// This value holds the address of the string containing the name of the function
// in which the failure occurred. This address value is not useful for anything
// other than helping the vendor to know in which file the failure  occurred.
EXTERN BOOL g_inFailureMode;  // Indicates that the TPM is in failure mode
#  if ALLOW_FORCE_FAILURE_MODE
EXTERN BOOL g_forceFailureMode;  // flag to force failure mode during test
#  endif

#  if FAIL_TRACE
// The name of the function that triggered failure mode.
EXTERN const char* s_failFunctionName;
#  endif  // FAIL_TRACE
// A numeric indicator of the function that triggered failure mode.
EXTERN UINT32 s_failFunction;
// The line in the file at which the error was signaled.
EXTERN UINT32 s_failLine;
// the reason for the failure.
EXTERN UINT32 s_failCode;

//**********************************************************************
//*** From ACT_spt.c
//**********************************************************************
// This value is used to indicate if an ACT has been updated since the last
// TPM2_Startup() (one bit for each ACT). If the ACT is not updated
// (TPM2_ACT_SetTimeout()) after a startup, then on each TPM2_Shutdown() the TPM will
// save 1/2 of the current timer value. This prevents an attack on the ACT by saving
// the counter and then running for a long period of time before doing a TPM Restart.
// A quick TPM2_Shutdown() after each
EXTERN UINT16 s_ActUpdated;

//**********************************************************************
//*** From CommandCodeAttributes.c
//**********************************************************************
// This array is instanced in CommandCodeAttributes.c when it includes
// CommandCodeAttributes.h. Don't change the extern to EXTERN.
extern const TPMA_CC            s_ccAttr[];
extern const COMMAND_ATTRIBUTES s_commandAttributes[];

#endif  // GLOBAL_H
```

6.24 /tpm/include/private/HandleProcess.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT
// clang-format off

#if CC_Startup
case TPM_CC_Startup:
    break;
#endif // CC_Startup
#if CC_Shutdown
case TPM_CC_Shutdown:
    break;
#endif // CC_Shutdown
#if CC_SelfTest
case TPM_CC_SelfTest:
    break;
#endif // CC_SelfTest
#if CC_IncrementalSelfTest
case TPM_CC_IncrementalSelfTest:
    break;
#endif // CC_IncrementalSelfTest
#if CC_GetTestResult
case TPM_CC_GetTestResult:
    break;
#endif // CC_GetTestResult
#if CC_StartAuthSession
case TPM_CC_StartAuthSession:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_ENTITY_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_StartAuthSession
#if CC_PolicyRestart
case TPM_CC_PolicyRestart:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyRestart
#if CC_Create
case TPM_CC_Create:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Create
#if CC_Load
case TPM_CC_Load:
```

```
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Load
#if CC_LoadExternal
case TPM_CC_LoadExternal:
    break;
#endif // CC_LoadExternal
#if CC_ReadPublic
case TPM_CC_ReadPublic:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ReadPublic
#if CC_ActivateCredential
case TPM_CC_ActivateCredential:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_ActivateCredential
#if CC_MakeCredential
case TPM_CC_MakeCredential:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_MakeCredential
#if CC_Unseal
case TPM_CC_Unseal:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Unseal
#if CC_ObjectChangeAuth
case TPM_CC_ObjectChangeAuth:
    *handleCount = 2;
```

```
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_ObjectChangeAuth
#if CC_CreateLoaded
case TPM_CC_CreateLoaded:
    *handleCount = 1;
    result = TPMI_DH_PARENT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_CreateLoaded
#if CC_Duplicate
case TPM_CC_Duplicate:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_Duplicate
#if CC_Rewrap
case TPM_CC_Rewrap:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_Rewrap
#if CC_Import
case TPM_CC_Import:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
```

```
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Import
#if CC_RSA_Encrypt
case TPM_CC_RSA_Encrypt:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_RSA_Encrypt
#if CC_RSA_Decrypt
case TPM_CC_RSA_Decrypt:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_RSA_Decrypt
#if CC_ECDH_KeyGen
case TPM_CC_ECDH_KeyGen:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ECDH_KeyGen
#if CC_ECDH_ZGen
case TPM_CC_ECDH_ZGen:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ECDH_ZGen
#if CC_ECC_Parameters
case TPM_CC_ECC_Parameters:
    break;
#endif // CC_ECC_Parameters
#if CC_ZGen_2Phase
case TPM_CC_ZGen_2Phase:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ZGen_2Phase
#if CC_ECC_Encrypt
case TPM_CC_ECC_Encrypt:
    *handleCount = 1;
```

```
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ECC_Encrypt
#if CC_ECC_Decrypt
case TPM_CC_ECC_Decrypt:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ECC_Decrypt
#if CC_EncryptDecrypt
case TPM_CC_EncryptDecrypt:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_EncryptDecrypt
#if CC_EncryptDecrypt2
case TPM_CC_EncryptDecrypt2:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_EncryptDecrypt2
#if CC_Hash
case TPM_CC_Hash:
    break;
#endif // CC_Hash
#if CC_HMAC
case TPM_CC_HMAC:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_HMAC
#if CC_MAC
case TPM_CC_MAC:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
```

```
#endif // CC_MAC
#if CC_GetRandom
case TPM_CC_GetRandom:
    break;
#endif // CC_GetRandom
#if CC_StirRandom
case TPM_CC_StirRandom:
    break;
#endif // CC_StirRandom
#if CC_HMAC_Start
case TPM_CC_HMAC_Start:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_HMAC_Start
#if CC_MAC_Start
case TPM_CC_MAC_Start:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_MAC_Start
#if CC_HashSequenceStart
case TPM_CC_HashSequenceStart:
    break;
#endif // CC_HashSequenceStart
#if CC_SequenceUpdate
case TPM_CC_SequenceUpdate:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_SequenceUpdate
#if CC_SequenceComplete
case TPM_CC_SequenceComplete:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_SequenceComplete
#if CC_EventSequenceComplete
case TPM_CC_EventSequenceComplete:
    *handleCount = 2;
    result = TPMI_DH_PCR_Unmarshal(&handles[0], handleBufferStart,
                                   bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
```

```
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_EventSequenceComplete
#if CC_Certify
case TPM_CC_Certify:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_Certify
#if CC_CertifyCreation
case TPM_CC_CertifyCreation:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_CertifyCreation
#if CC_Quote
case TPM_CC_Quote:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Quote
#if CC_GetSessionAuditDigest
case TPM_CC_GetSessionAuditDigest:
    *handleCount = 3;
    result = TPMI_RH_ENDORSEMENT_Unmarshal(&handles[0], handleBufferStart,
                                           bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
```

```
        return result + TPM_RC_H + TPM_RC_2;
    }
    result = TPMI_SH_HMAC_Unmarshal(&handles[2], handleBufferStart,
                                    bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_3;
    }
    break;
#endif // CC_GetSessionAuditDigest
#if CC_GetCommandAuditDigest
case TPM_CC_GetCommandAuditDigest:
    *handleCount = 2;
    result = TPMI_RH_ENDORSEMENT_Unmarshal(&handles[0], handleBufferStart,
                                           bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_GetCommandAuditDigest
#if CC_GetTime
case TPM_CC_GetTime:
    *handleCount = 2;
    result = TPMI_RH_ENDORSEMENT_Unmarshal(&handles[0], handleBufferStart,
                                           bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_GetTime
#if CC_CertifyX509
case TPM_CC_CertifyX509:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_CertifyX509
#if CC_Commit
case TPM_CC_Commit:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
```

```
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Commit
#if CC_EC_Ephemeral
case TPM_CC_EC_Ephemeral:
    break;
#endif // CC_EC_Ephemeral
#if CC_VerifySignature
case TPM_CC_VerifySignature:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_VerifySignature
#if CC_Sign
case TPM_CC_Sign:
    *handleCount = 1;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Sign
#if CC_SetCommandCodeAuditStatus
case TPM_CC_SetCommandCodeAuditStatus:
    *handleCount = 1;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_SetCommandCodeAuditStatus
#if CC_PCR_Extend
case TPM_CC_PCR_Extend:
    *handleCount = 1;
    result = TPMI_DH_PCR_Unmarshal(&handles[0], handleBufferStart,
                                   bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PCR_Extend
#if CC_PCR_Event
case TPM_CC_PCR_Event:
    *handleCount = 1;
    result = TPMI_DH_PCR_Unmarshal(&handles[0], handleBufferStart,
                                   bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PCR_Event
#if CC_PCR_Read
```

```
case TPM_CC_PCR_Read:
    break;
#endif // CC_PCR_Read
#if CC_PCR_Allocate
case TPM_CC_PCR_Allocate:
    *handleCount = 1;
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PCR_Allocate
#if CC_PCR_SetAuthPolicy
case TPM_CC_PCR_SetAuthPolicy:
    *handleCount = 1;
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PCR_SetAuthPolicy
#if CC_PCR_SetAuthValue
case TPM_CC_PCR_SetAuthValue:
    *handleCount = 1;
    result = TPMI_DH_PCR_Unmarshal(&handles[0], handleBufferStart,
                                   bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PCR_SetAuthValue
#if CC_PCR_Reset
case TPM_CC_PCR_Reset:
    *handleCount = 1;
    result = TPMI_DH_PCR_Unmarshal(&handles[0], handleBufferStart,
                                   bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PCR_Reset
#if CC_PolicySigned
case TPM_CC_PolicySigned:
    *handleCount = 2;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_SH_POLICY_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_PolicySigned
#if CC_PolicySecret
case TPM_CC_PolicySecret:
```

```
    *handleCount = 2;
    result = TPMI_DH_ENTITY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_SH_POLICY_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_PolicySecret
#if CC_PolicyTicket
case TPM_CC_PolicyTicket:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyTicket
#if CC_PolicyOR
case TPM_CC_PolicyOR:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyOR
#if CC_PolicyPCR
case TPM_CC_PolicyPCR:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyPCR
#if CC_PolicyLocality
case TPM_CC_PolicyLocality:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyLocality
#if CC_PolicyNV
case TPM_CC_PolicyNV:
    *handleCount = 3;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
```

```
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    result = TPMI_SH_POLICY_Unmarshal(&handles[2], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_3;
    }
    break;
#endif // CC_PolicyNV
#if CC_PolicyCounterTimer
case TPM_CC_PolicyCounterTimer:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyCounterTimer
#if CC_PolicyCommandCode
case TPM_CC_PolicyCommandCode:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyCommandCode
#if CC_PolicyPhysicalPresence
case TPM_CC_PolicyPhysicalPresence:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyPhysicalPresence
#if CC_PolicyCpHash
case TPM_CC_PolicyCpHash:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyCpHash
#if CC_PolicyNameHash
case TPM_CC_PolicyNameHash:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
```

```
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyNameHash
#if CC_PolicyDuplicationSelect
case TPM_CC_PolicyDuplicationSelect:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyDuplicationSelect
#if CC_PolicyAuthorize
case TPM_CC_PolicyAuthorize:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyAuthorize
#if CC_PolicyAuthValue
case TPM_CC_PolicyAuthValue:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyAuthValue
#if CC_PolicyPassword
case TPM_CC_PolicyPassword:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyPassword
#if CC_PolicyGetDigest
case TPM_CC_PolicyGetDigest:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyGetDigest
#if CC_PolicyNvWritten
case TPM_CC_PolicyNvWritten:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
```

```
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyNvWritten
#if CC_PolicyTemplate
case TPM_CC_PolicyTemplate:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyTemplate
#if CC_PolicyAuthorizeNV
case TPM_CC_PolicyAuthorizeNV:
    *handleCount = 3;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    result = TPMI_SH_POLICY_Unmarshal(&handles[2], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_3;
    }
    break;
#endif // CC_PolicyAuthorizeNV
#if CC_PolicyCapability
case TPM_CC_PolicyCapability:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyCapability
#if CC_PolicyParameters
case TPM_CC_PolicyParameters:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PolicyParameters
#if CC_CreatePrimary
case TPM_CC_CreatePrimary:
    *handleCount = 1;
    result = TPMI_RH_HIERARCHY_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
```

```
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_CreatePrimary
#if CC_HierarchyControl
case TPM_CC_HierarchyControl:
    *handleCount = 1;
    result = TPMI_RH_BASE_HIERARCHY_Unmarshal(&handles[0], handleBufferStart,
                                              bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_HierarchyControl
#if CC_SetPrimaryPolicy
case TPM_CC_SetPrimaryPolicy:
    *handleCount = 1;
    result = TPMI_RH_HIERARCHY_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                                bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_SetPrimaryPolicy
#if CC_ChangePPS
case TPM_CC_ChangePPS:
    *handleCount = 1;
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ChangePPS
#if CC_ChangeEPS
case TPM_CC_ChangeEPS:
    *handleCount = 1;
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ChangeEPS
#if CC_Clear
case TPM_CC_Clear:
    *handleCount = 1;
    result = TPMI_RH_CLEAR_Unmarshal(&handles[0], handleBufferStart,
                                     bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Clear
#if CC_ClearControl
case TPM_CC_ClearControl:
    *handleCount = 1;
    result = TPMI_RH_CLEAR_Unmarshal(&handles[0], handleBufferStart,
                                     bufferRemainingSize);
```

```
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ClearControl
#if CC_HierarchyChangeAuth
case TPM_CC_HierarchyChangeAuth:
    *handleCount = 1;
    result = TPMI_RH_HIERARCHY_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                              bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_HierarchyChangeAuth
#if CC_DictionaryAttackLockReset
case TPM_CC_DictionaryAttackLockReset:
    *handleCount = 1;
    result = TPMI_RH_LOCKOUT_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_DictionaryAttackLockReset
#if CC_DictionaryAttackParameters
case TPM_CC_DictionaryAttackParameters:
    *handleCount = 1;
    result = TPMI_RH_LOCKOUT_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_DictionaryAttackParameters
#if CC_PP_Commands
case TPM_CC_PP_Commands:
    *handleCount = 1;
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_PP_Commands
#if CC_SetAlgorithmSet
case TPM_CC_SetAlgorithmSet:
    *handleCount = 1;
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_SetAlgorithmSet
#if CC_FieldUpgradeStart
case TPM_CC_FieldUpgradeStart:
    *handleCount = 2;
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
```

```
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_FieldUpgradeStart
#if CC_FieldUpgradeData
case TPM_CC_FieldUpgradeData:
    break;
#endif // CC_FieldUpgradeData
#if CC_FirmwareRead
case TPM_CC_FirmwareRead:
    break;
#endif // CC_FirmwareRead
#if CC_ContextSave
case TPM_CC_ContextSave:
    *handleCount = 1;
    result = TPMI_DH_CONTEXT_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ContextSave
#if CC_ContextLoad
case TPM_CC_ContextLoad:
    break;
#endif // CC_ContextLoad
#if CC_FlushContext
case TPM_CC_FlushContext:
    break;
#endif // CC_FlushContext
#if CC_EvictControl
case TPM_CC_EvictControl:
    *handleCount = 2;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_DH_OBJECT_Unmarshal(&handles[1], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_EvictControl
#if CC_ReadClock
case TPM_CC_ReadClock:
    break;
#endif // CC_ReadClock
#if CC_ClockSet
case TPM_CC_ClockSet:
    *handleCount = 1;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
```

```
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ClockSet
#if CC_ClockRateAdjust
case TPM_CC_ClockRateAdjust:
    *handleCount = 1;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_ClockRateAdjust
#if CC_GetCapability
case TPM_CC_GetCapability:
    break;
#endif // CC_GetCapability
#if CC_TestParms
case TPM_CC_TestParms:
    break;
#endif // CC_TestParms
#if CC_NV_DefineSpace
case TPM_CC_NV_DefineSpace:
    *handleCount = 1;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_NV_DefineSpace
#if CC_NV_UndefineSpace
case TPM_CC_NV_UndefineSpace:
    *handleCount = 2;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_DEFINED_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                                bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_UndefineSpace
#if CC_NV_UndefineSpaceSpecial
case TPM_CC_NV_UndefineSpaceSpecial:
    *handleCount = 2;
    result = TPMI_RH_NV_DEFINED_INDEX_Unmarshal(&handles[0], handleBufferStart,
                                                bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_PLATFORM_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
```

```
    }
    break;
#endif // CC_NV_UndefineSpaceSpecial
#if CC_NV_ReadPublic
case TPM_CC_NV_ReadPublic:
    *handleCount = 1;
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_NV_ReadPublic
#if CC_NV_Write
case TPM_CC_NV_Write:
    *handleCount = 2;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_Write
#if CC_NV_Increment
case TPM_CC_NV_Increment:
    *handleCount = 2;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_Increment
#if CC_NV_Extend
case TPM_CC_NV_Extend:
    *handleCount = 2;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_Extend
#if CC_NV_SetBits
```

```
case TPM_CC_NV_SetBits:
    *handleCount = 2;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_SetBits
#if CC_NV_WriteLock
case TPM_CC_NV_WriteLock:
    *handleCount = 2;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_WriteLock
#if CC_NV_GlobalWriteLock
case TPM_CC_NV_GlobalWriteLock:
    *handleCount = 1;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_NV_GlobalWriteLock
#if CC_NV_Read
case TPM_CC_NV_Read:
    *handleCount = 2;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_Read
#if CC_NV_ReadLock
case TPM_CC_NV_ReadLock:
    *handleCount = 2;
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[0], handleBufferStart,
                                       bufferRemainingSize);
```

```
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[1], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    break;
#endif // CC_NV_ReadLock
#if CC_NV_ChangeAuth
case TPM_CC_NV_ChangeAuth:
    *handleCount = 1;
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_NV_ChangeAuth
#if CC_NV_Certify
case TPM_CC_NV_Certify:
    *handleCount = 3;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[1], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[2], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_3;
    }
    break;
#endif // CC_NV_Certify
#if CC_NV_DefineSpace2
case TPM_CC_NV_DefineSpace2:
    *handleCount = 1;
    result = TPMI_RH_PROVISION_Unmarshal(&handles[0], handleBufferStart,
                                         bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_NV_DefineSpace2
#if CC_NV_ReadPublic2
case TPM_CC_NV_ReadPublic2:
    *handleCount = 1;
    result = TPMI_RH_NV_INDEX_Unmarshal(&handles[0], handleBufferStart,
                                        bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
```

```
    break;
#endif // CC_NV_ReadPublic2
#if CC_SetCapability
case TPM_CC_SetCapability:
    *handleCount = 1;
    result = TPMI_RH_HIERARCHY_UNMARSHAL(&handles[0], handleBufferStart,
                                         bufferRemainingSize, TRUE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_SetCapability
#if CC_AC_GetCapability
case TPM_CC_AC_GetCapability:
    *handleCount = 1;
    result = TPMI_RH_AC_Unmarshal(&handles[0], handleBufferStart,
                                  bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_AC_GetCapability
#if CC_AC_Send
case TPM_CC_AC_Send:
    *handleCount = 3;
    result = TPMI_DH_OBJECT_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize, FALSE);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    result = TPMI_RH_NV_AUTH_Unmarshal(&handles[1], handleBufferStart,
                                       bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_2;
    }
    result = TPMI_RH_AC_Unmarshal(&handles[2], handleBufferStart,
                                  bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_3;
    }
    break;
#endif // CC_AC_Send
#if CC_Policy_AC_SendSelect
case TPM_CC_Policy_AC_SendSelect:
    *handleCount = 1;
    result = TPMI_SH_POLICY_Unmarshal(&handles[0], handleBufferStart,
                                      bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
    }
    break;
#endif // CC_Policy_AC_SendSelect
#if CC_ACT_SetTimeout
case TPM_CC_ACT_SetTimeout:
    *handleCount = 1;
    result = TPMI_RH_ACT_Unmarshal(&handles[0], handleBufferStart,
                                   bufferRemainingSize);
    if (TPM_RC_SUCCESS != result)
    {
        return result + TPM_RC_H + TPM_RC_1;
```

```
    }
    break;
#endif // CC_ACT_SetTimeout
#if CC_Vendor_TCG_Test
case TPM_CC_Vendor_TCG_Test:
    break;
#endif // CC_Vendor_TCG_Test
```

6.25 /tpm/include/private/HashTestData.h

```
//
// Hash Test Vectors
//

TPM2B_TYPE(HASH_TEST_KEY, 128);  // Twice the largest digest size
TPM2B_HASH_TEST_KEY c_hashTestKey = {
    {128,
     {0xa0, 0xed, 0x5c, 0x9a, 0xd2, 0x4a, 0x21, 0x40, 0x1a, 0xd0, 0x81, 0x47, 0x39,
      0x63, 0xf9, 0x50, 0xdc, 0x59, 0x47, 0x11, 0x40, 0x13, 0x99, 0x92, 0xc0, 0x72,
      0xa4, 0x0f, 0xe2, 0x33, 0xe4, 0x63, 0x9b, 0xb6, 0x76, 0xc3, 0x1e, 0x6f, 0x13,
      0xee, 0xcc, 0x99, 0x71, 0xa5, 0xc0, 0xcf, 0x9a, 0x40, 0xcf, 0xdb, 0x66, 0x70,
      0x05, 0x63, 0x54, 0x12, 0x25, 0xf4, 0xe0, 0x1b, 0x23, 0x35, 0xe3, 0x70, 0x7d,
      0x19, 0x5f, 0x00, 0xe4, 0xf1, 0x61, 0x73, 0x05, 0xd8, 0x58, 0x7f, 0x60, 0x61,
      0x84, 0x36, 0xec, 0xbe, 0x96, 0x1b, 0x69, 0x00, 0xf0, 0x9a, 0x6e, 0xe3, 0x26,
      0x73, 0x0d, 0x17, 0x5b, 0x33, 0x41, 0x44, 0x9d, 0x90, 0xab, 0xd9, 0x6b, 0x7d,
      0x48, 0x99, 0x25, 0x93, 0x29, 0x14, 0x2b, 0xce, 0x93, 0x8d, 0x8c, 0xaf, 0x31,
      0x0e, 0x9c, 0x57, 0xd8, 0x5b, 0x57, 0x20, 0x1b, 0x9f, 0x2d, 0xa5}}};

TPM2B_TYPE(HASH_TEST_DATA, 256);  // Twice the largest block size
TPM2B_HASH_TEST_DATA c_hashTestData = {
    {256,
     {0x88, 0xac, 0xc3, 0xe5, 0x5f, 0x66, 0x9d, 0x18, 0x80, 0xc9, 0x7a, 0x9c, 0xa4,
      0x08, 0x90, 0x98, 0x0f, 0x3a, 0x53, 0x92, 0x4c, 0x67, 0x4e, 0xb7, 0x37, 0xec,
      0x67, 0x87, 0xb6, 0xbe, 0x10, 0xca, 0x11, 0x5b, 0x4a, 0x0b, 0x45, 0xc3, 0x32,
      0x68, 0x48, 0x69, 0xce, 0x25, 0x1b, 0xc8, 0xaf, 0x44, 0x79, 0x22, 0x83, 0xc8,
      0xfb, 0xe2, 0x63, 0x94, 0xa2, 0x3c, 0x59, 0x3e, 0x3e, 0xc6, 0x64, 0x2c, 0x1f,
      0x8c, 0x11, 0x93, 0x24, 0xa3, 0x17, 0xc5, 0x2f, 0x37, 0xcf, 0x95, 0x97, 0x8e,
      0x63, 0x39, 0x68, 0xd5, 0xca, 0xba, 0x18, 0x37, 0x69, 0x6e, 0x4f, 0x19, 0xfd,
      0x8a, 0xc0, 0x8d, 0x87, 0x3a, 0xbc, 0x31, 0x42, 0x04, 0x05, 0xef, 0xb5, 0x02,
      0xef, 0x1e, 0x92, 0x4b, 0xb7, 0x73, 0x2c, 0x8c, 0xeb, 0x23, 0x13, 0x81, 0x34,
      0xb9, 0xb5, 0xc1, 0x17, 0x37, 0x39, 0xf8, 0x3e, 0xe4, 0x4c, 0x06, 0xa8, 0x81,
      0x52, 0x2f, 0xef, 0xc9, 0x9c, 0x69, 0x89, 0xbc, 0x85, 0x9c, 0x30, 0x16, 0x02,
      0xca, 0xe3, 0x61, 0xd4, 0x0f, 0xed, 0x34, 0x1b, 0xca, 0xc1, 0x1b, 0xd1, 0xfa,
      0xc1, 0xa2, 0xe0, 0xdf, 0x52, 0x2f, 0x0b, 0x4b, 0x9f, 0x0e, 0x45, 0x54, 0xb9,
      0x17, 0xb6, 0xaf, 0xd6, 0xd5, 0xca, 0x90, 0x29, 0x57, 0x7b, 0x70, 0x50, 0x94,
      0x5c, 0x8e, 0xf6, 0x4e, 0x21, 0x8b, 0xc6, 0x8b, 0xa6, 0xbc, 0xb9, 0x64, 0xd4,
      0x4d, 0xf3, 0x68, 0xd8, 0xac, 0xde, 0xd8, 0xd8, 0xb5, 0x6d, 0xcd, 0x93, 0xeb,
      0x28, 0xa4, 0xe2, 0x5c, 0x44, 0xef, 0xf0, 0xe1, 0x6f, 0x38, 0x1a, 0x3c, 0xe6,
      0xef, 0xa2, 0x9d, 0xb9, 0xa8, 0x05, 0x2a, 0x95, 0xec, 0x5f, 0xdb, 0xb0, 0x25,
      0x67, 0x9c, 0x86, 0x7a, 0x8e, 0xea, 0x51, 0xcc, 0xc3, 0xd3, 0xff, 0x6e, 0xf0,
      0xed, 0xa3, 0xae, 0xf9, 0x5d, 0x33, 0x70, 0xf2, 0x11}}};

#if ALG_SHA1 == YES
TPM2B_TYPE(SHA1, 20);
TPM2B_SHA1 c_SHA1_digest = {
    {20, {0xee, 0x2c, 0xef, 0x93, 0x76, 0xbd, 0xf8, 0x91, 0xbc, 0xe6,
          0xe5, 0x57, 0x53, 0x77, 0x01, 0xb5, 0x70, 0x95, 0xe5, 0x40}}};
#endif

#if ALG_SHA256 == YES
TPM2B_TYPE(SHA256, 32);
TPM2B_SHA256 c_SHA256_digest = {
    {32, {0x64, 0xe8, 0xe0, 0xc3, 0xa9, 0xa4, 0x51, 0x49, 0x10, 0x55, 0x8d,
          0x31, 0x71, 0xe5, 0x2f, 0x69, 0x3a, 0xdc, 0xc7, 0x11, 0x32, 0x44,
          0x61, 0xbd, 0x34, 0x39, 0x57, 0xb0, 0xa8, 0x75, 0x86, 0x1b}}};
```

```
#endif

#if ALG_SHA384 == YES
TPM2B_TYPE(SHA384, 48);
TPM2B_SHA384 c_SHA384_digest = {
    {48, {0x37, 0x75, 0x29, 0xb5, 0x20, 0x15, 0x6e, 0xa3, 0x7e, 0xa3, 0x0d, 0xcd,
          0x80, 0xa8, 0xa3, 0x3d, 0xeb, 0xe8, 0xad, 0x4e, 0x1c, 0x77, 0x94, 0x5a,
          0xaf, 0x6c, 0xd0, 0xc1, 0xfa, 0x43, 0x3f, 0xc7, 0xb8, 0xf1, 0x01, 0xc0,
          0x60, 0xbf, 0xf2, 0x87, 0xe8, 0x71, 0x9e, 0x51, 0x97, 0xa0, 0x09, 0x8d}}};
#endif

#if ALG_SHA512 == YES
TPM2B_TYPE(SHA512, 64);
TPM2B_SHA512 c_SHA512_digest = {
    {64,
     {0xe2, 0x7b, 0x10, 0x3d, 0x5e, 0x48, 0x58, 0x44, 0x67, 0xac, 0xa3, 0x81, 0x8c,
      0x1d, 0xc5, 0x71, 0x66, 0x92, 0x8a, 0x89, 0xaa, 0xd4, 0x35, 0x51, 0x60, 0x37,
      0x31, 0xd7, 0xba, 0xe7, 0x93, 0x0b, 0x16, 0x4d, 0xb3, 0xc8, 0x34, 0x98, 0x3c,
      0xd3, 0x53, 0xde, 0x5e, 0xe8, 0x0c, 0xbc, 0xaf, 0xc9, 0x24, 0x2c, 0xcc, 0xed,
      0xdb, 0xde, 0xba, 0x1f, 0x14, 0x14, 0x5a, 0x95, 0x80, 0xde, 0x66, 0xbd}}};
#endif

TPM2B_TYPE(EMPTY, 1);

#if ALG_SM3_256 == YES
TPM2B_EMPTY c_SM3_256_digest = {{0, {0}}};
#endif

#if ALG_SHA3_256 == YES
TPM2B_EMPTY c_SHA3_256_digest = {{0, {0}}};
#endif

#if ALG_SHA3_384 == YES
TPM2B_EMPTY c_SHA3_384_digest = {{0, {0}}};
#endif

#if ALG_SHA3_512 == YES
TPM2B_EMPTY c_SHA3_512_digest = {{0, {0}}};
#endif
```

6.26 /tpm/include/private/InternalRoutines.h

```
#ifndef INTERNAL_ROUTINES_H
#define INTERNAL_ROUTINES_H

#if !defined _LIB_SUPPORT_H_ && !defined _TPM_H_
#  error "Should not be called"
#endif

// DRTM functions
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/prototypes/_TPM_Hash_Start_fp.h>
#include <platform_interface/prototypes/_TPM_Hash_Data_fp.h>
#include <platform_interface/prototypes/_TPM_Hash_End_fp.h>

// Internal subsystem functions
#include "Object_fp.h"
#include "Context_spt_fp.h"
#include "Object_spt_fp.h"
#include "Entity_fp.h"
#include "Session_fp.h"
#include "Hierarchy_fp.h"
#include "NvReserved_fp.h"
#include "NvDynamic_fp.h"
```

```
#include "NV_spt_fp.h"
#include "ACT_spt_fp.h"
#include "PCR_fp.h"
#include "DA_fp.h"
// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/prototypes/TpmFail_fp.h>
#include "SessionProcess_fp.h"

// Internal support functions
#include "CommandCodeAttributes_fp.h"
#include "Marshal.h"
#include "Time_fp.h"
#include "Locality_fp.h"
#include "PP_fp.h"
#include "CommandAudit_fp.h"
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/prototypes/Manufacture_fp.h>
#include "Handle_fp.h"
#include "Power_fp.h"
#include "Response_fp.h"
#include "CommandDispatcher_fp.h"

#ifdef CC_AC_Send
#  include "AC_spt_fp.h"
#endif  // CC_AC_Send

// Miscellaneous
#include "Bits_fp.h"
#include "AlgorithmCap_fp.h"
#include "PropertyCap_fp.h"
#include "IoBuffers_fp.h"
#include "Memory_fp.h"
#include "ResponseCodeProcessing_fp.h"

// Asymmetric Support library Interface
// TODO_RENAME_INC_FOLDER: needs a component prefix
// Math interface must be included before other Crypt headers to define types
#include <MathLibraryInterface.h>

// Internal cryptographic functions
#include "Ticket_fp.h"
#include "CryptUtil_fp.h"
#include "CryptHash_fp.h"
#include "CryptSym_fp.h"
#include "CryptPrime_fp.h"
#include "CryptRand_fp.h"
#include "CryptSelfTest_fp.h"
#include "MathOnByteBuffers_fp.h"
#include "CryptSym_fp.h"
#include "AlgorithmTests_fp.h"

#if ALG_RSA
#  include "CryptRsa_fp.h"
#  include "CryptPrimeSieve_fp.h"
#endif

#if ALG_ECC
#  include "CryptEccMain_fp.h"
#  include "CryptEccSignature_fp.h"
#  include "CryptEccKeyExchange_fp.h"
#  include "CryptEccCrypt_fp.h"
#endif

#if CC_MAC || CC_MAC_Start
#  include "CryptSmac_fp.h"
```

```
#  if ALG_CMAC
#    include "CryptCmac_fp.h"
#  endif
#endif

// Linkage to platform functions
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/tpm_to_platform_interface.h>

#endif
```

6.27 /tpm/include/private/KdfTestData.h

```
//
// Hash Test Vectors
//

#define TEST_KDF_KEY_SIZE 20

TPM2B_TYPE(KDF_TEST_KEY, TEST_KDF_KEY_SIZE);
TPM2B_KDF_TEST_KEY c_kdfTestKeyIn = {
    {TEST_KDF_KEY_SIZE,
     {0x27, 0x1F, 0xA0, 0x8B, 0xBD, 0xC5, 0x06, 0x0E, 0xC3, 0xDF,
      0xA9, 0x28, 0xFF, 0x9B, 0x73, 0x12, 0x3A, 0x12, 0xDA, 0x0C}}};

TPM2B_TYPE(KDF_TEST_LABEL, 17);
TPM2B_KDF_TEST_LABEL c_kdfTestLabel = {{17,
                                        {0x4B,
                                         0x44,
                                         0x46,
                                         0x53,
                                         0x45,
                                         0x4C,
                                         0x46,
                                         0x54,
                                         0x45,
                                         0x53,
                                         0x54,
                                         0x4C,
                                         0x41,
                                         0x42,
                                         0x45,
                                         0x4C,
                                         0x00}}};

TPM2B_TYPE(KDF_TEST_CONTEXT, 8);
TPM2B_KDF_TEST_CONTEXT c_kdfTestContextU = {
    {8, {0xCE, 0x24, 0x4F, 0x39, 0x5D, 0xCA, 0x73, 0x91}}};

TPM2B_KDF_TEST_CONTEXT c_kdfTestContextV = {
    {8, {0xDA, 0x50, 0x40, 0x31, 0xDD, 0xF1, 0x2E, 0x83}}};

#if ALG_SHA512 == ALG_YES
TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {
    {20, {0x8b, 0xe2, 0xc1, 0xb8, 0x5b, 0x78, 0x56, 0x9b, 0x9f, 0xa7,
          0x59, 0xf5, 0x85, 0x7c, 0x56, 0xd6, 0x84, 0x81, 0x0f, 0xd3}}};
#  define KDF_TEST_ALG TPM_ALG_SHA512

#elif ALG_SHA384 == ALG_YES
TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {
    {20, {0x1d, 0xce, 0x70, 0xc9, 0x11, 0x3e, 0xb2, 0xdb, 0xa4, 0x7b,
          0xd9, 0xcf, 0xc7, 0x2b, 0xf4, 0x6f, 0x45, 0xb0, 0x93, 0x12}}};
#  define KDF_TEST_ALG TPM_ALG_SHA384
```

```
#elif ALG_SHA256 == ALG_YES
TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {
    {20, {0xbb, 0x02, 0x59, 0xe1, 0xc8, 0xba, 0x60, 0x7e, 0x6a, 0x2c,
          0xd7, 0x04, 0xb6, 0x9a, 0x90, 0x2e, 0x9a, 0xde, 0x84, 0xc4}}};
#  define KDF_TEST_ALG TPM_ALG_SHA256

#elif ALG_SHA1 == ALG_YES
TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {
    {20, {0x55, 0xb5, 0xa7, 0x18, 0x4a, 0xa0, 0x74, 0x23, 0xc4, 0x7d,
          0xae, 0x76, 0x6c, 0x26, 0xa2, 0x37, 0x7d, 0x7c, 0xf8, 0x51}}};
#  define KDF_TEST_ALG TPM_ALG_SHA1
#endif
```

6.28 /tpm/include/private/LibSupport.h

```
// This header file is used to select the library code that gets included in the
// TPM build.

#ifndef _LIB_SUPPORT_H_
#define _LIB_SUPPORT_H_
// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/tpm_radix.h>

// Include the options for hashing and symmetric. Defer the load of the math package
// Until the bignum parameters are defined.
#ifndef SYM_LIB
#  error SYM_LIB required
#endif
#ifndef HASH_LIB
#  error HASH_LIB required
#endif

#include LIB_INCLUDE(TpmTo, SYM_LIB, Sym)
#include LIB_INCLUDE(TpmTo, HASH_LIB, Hash)

//TODO: was #undef MIN
//was #undef MAX

#endif  // _LIB_SUPPORT_H_
```

6.29 /tpm/include/private/Marshal.h

```
//** Introduction
// This file is used to provide the things needed by a module that uses the marshaling
// functions. It handles the variations between the marshaling choices (procedural or
// table-driven).

#if TABLE_DRIVEN_MARSHAL

#  include "TableMarshalTypes.h"

#  include "TableMarshalDefines.h"

#  include "TableDrivenMarshal_fp.h"

#else

#  include "Marshal_fp.h"

#endif
```

6.30 /tpm/include/private/NV.h

```
//** Index Type Definitions

// These definitions allow the same code to be used pre and post 1.21. The main
// action is to redefine the index type values from the bit values.
// Use TPM_NT_ORDINARY to indicate if the TPM_NT type is defined

#ifndef _NV_H_
#define _NV_H_

#ifdef TPM_NT_ORDINARY
// If TPM_NT_ORDINARY is defined, then the TPM_NT field is present in a TPMA_NV
#  define GET_TPM_NT(attributes) GET_ATTRIBUTE(attributes, TPMA_NV, TPM_NT)
#else
// If TPM_NT_ORDINARY is not defined, then need to synthesize it from the
// attributes
#  define GetNv_TPM_NV(attributes)                        \
      (IS_ATTRIBUTE(attributes, TPMA_NV, COUNTER)         \
       + (IS_ATTRIBUTE(attributes, TPMA_NV, BITS) << 1)  \
       + (IS_ATTRIBUTE(attributes, TPMA_NV, EXTEND) << 2))
#  define TPM_NT_ORDINARY (0)
#  define TPM_NT_COUNTER  (1)
#  define TPM_NT_BITS     (2)
#  define TPM_NT_EXTEND   (4)
#endif

//** Attribute Macros
// These macros are used to isolate the differences in the way that the index type
// changed in version 1.21 of the specification
#define IsNvOrdinaryIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_ORDINARY)

#define IsNvCounterIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_COUNTER)

#define IsNvBitsIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_BITS)

#define IsNvExtendIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_EXTEND)

#ifdef TPM_NT_PIN_PASS
#  define IsNvPinPassIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_PIN_PASS)
#endif

#ifdef TPM_NT_PIN_FAIL
#  define IsNvPinFailIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_PIN_FAIL)
#endif

typedef struct
{
    UINT32     size;
    TPM_HANDLE handle;
} NV_ENTRY_HEADER;

#define NV_EVICT_OBJECT_SIZE (sizeof(UINT32) + sizeof(TPM_HANDLE) + sizeof(OBJECT))

#define NV_INDEX_COUNTER_SIZE (sizeof(UINT32) + sizeof(NV_INDEX) + sizeof(UINT64))

#define NV_RAM_INDEX_COUNTER_SIZE (sizeof(NV_RAM_HEADER) + sizeof(UINT64))

typedef struct
{
    UINT32     size;
    TPM_HANDLE handle;
    TPMA_NV    attributes;
} NV_RAM_HEADER;

// Defines the end-of-list marker for NV. The list terminator is
```

```
// a UINT32 of zero, followed by the current value of s_maxCounter which is a
// 64-bit value. The structure is defined as an array of 3 UINT32 values so that
// there is no padding between the  UINT32 list end marker and the UINT64 maxCounter
// value.
typedef UINT32 NV_LIST_TERMINATOR[3];

//** Orderly RAM Values
// The following defines are for accessing orderly RAM values.

// This is the initialize for the RAM reference iterator.
#define NV_RAM_REF_INIT 0
// This is the starting address of the RAM space used for orderly data
#define RAM_ORDERLY_START (&s_indexOrderlyRam[0])
// This is the offset within NV that is used to save the orderly data on an
// orderly shutdown.
#define NV_ORDERLY_START (NV_INDEX_RAM_DATA)
// This is the end of the orderly RAM space. It is actually the first byte after the
// last byte of orderly RAM data
#define RAM_ORDERLY_END (RAM_ORDERLY_START + sizeof(s_indexOrderlyRam))
// This is the end of the orderly space in NV memory. As with RAM_ORDERLY_END, it is
// actually the offset of the first byte after the end of the NV orderly data.
#define NV_ORDERLY_END (NV_ORDERLY_START + sizeof(s_indexOrderlyRam))

// Macro to check that an orderly RAM address is with range.
#define ORDERLY_RAM_ADDRESS_OK(start, offset) \
    ((start >= RAM_ORDERLY_START) && ((start + offset - 1) < RAM_ORDERLY_END))

#define RETURN_IF_NV_IS_NOT_AVAILABLE    \
    {                                    \
        if(g_NvStatus != TPM_RC_SUCCESS) \
            return g_NvStatus;           \
    }

// Routinely have to clear the orderly flag and fail if the
// NV is not available so that it can be cleared.
#define RETURN_IF_ORDERLY                          \
    {                                              \
        if(NvClearOrderly() != TPM_RC_SUCCESS)     \
            return g_NvStatus;                     \
    }

#define NV_IS_AVAILABLE (g_NvStatus == TPM_RC_SUCCESS)

#define IS_ORDERLY(value) (value < SU_DA_USED_VALUE)

#define NV_IS_ORDERLY (IS_ORDERLY(gp.orderlyState))

// Macro to set the NV UPDATE_TYPE. This deals with the fact that the update is
// possibly a combination of UT_NV and UT_ORDERLY.
#define SET_NV_UPDATE(type) g_updateNV |= (type)

#endif  // _NV_H_
```

6.31 /tpm/include/private/OIDs.h

```
#ifndef _OIDS_H_
#define _OIDS_H_

// All the OIDs in this file are defined as DER-encoded values with a leading tag
// 0x06 (ASN1_OBJECT_IDENTIFIER), followed by a single length byte. This allows the
// OID size to be determined by looking at octet[1] of the OID (total size is
// OID[1] + 2).

// These macros allow OIDs to be defined (or not) depending on whether the associated
// hash algorithm is implemented.
```

```
// NOTE: When one of these macros is used, the NAME needs '_" on each side. The
// exception is when the macro is used for the hash OID when only a single '_' is
// used.
#ifndef ALG_SHA1
#  define ALG_SHA1 NO
#endif
#if ALG_SHA1
#  define SHA1_OID(NAME) MAKE_OID(NAME##SHA1)
#else
#  define SHA1_OID(NAME)
#endif
#ifndef ALG_SHA256
#  define ALG_SHA256 NO
#endif
#if ALG_SHA256
#  define SHA256_OID(NAME) MAKE_OID(NAME##SHA256)
#else
#  define SHA256_OID(NAME)
#endif
#ifndef ALG_SHA384
#  define ALG_SHA384 NO
#endif
#if ALG_SHA384
#  define SHA384_OID(NAME) MAKE_OID(NAME##SHA384)
#else
#  define SHA384_OID(NAME)
#endif
#ifndef ALG_SHA512
#  define ALG_SHA512 NO
#endif
#if ALG_SHA512
#  define SHA512_OID(NAME) MAKE_OID(NAME##SHA512)
#else
#  define SHA512_OID(NAME)
#endif
#ifndef ALG_SM3_256
#  define ALG_SM3_256 NO
#endif
#if ALG_SM3_256
#  define SM3_256_OID(NAME) MAKE_OID(NAME##SM3_256)
#else
#  define SM3_256_OID(NAME)
#endif
#ifndef ALG_SHA3_256
#  define ALG_SHA3_256 NO
#endif
#if ALG_SHA3_256
#  define SHA3_256_OID(NAME) MAKE_OID(NAME##SHA3_256)
#else
#  define SHA3_256_OID(NAME)
#endif
#ifndef ALG_SHA3_384
#  define ALG_SHA3_384 NO
#endif
#if ALG_SHA3_384
#  define SHA3_384_OID(NAME) MAKE_OID(NAME##SHA3_384)
#else
#  define SHA3_384_OID(NAME)
#endif
#ifndef ALG_SHA3_512
#  define ALG_SHA3_512 NO
#endif
#if ALG_SHA3_512
#  define SHA3_512_OID(NAME) MAKE_OID(NAME##SHA3_512)
#else
#  define SHA3_512_OID(NAME)
```

```
#endif

// These are encoded to take one additional byte of algorithm selector
#define NIST_HASH 0x06, 0x09, 0x60, 0x86, 0x48, 1, 101, 3, 4, 2
#define NIST_SIG  0x06, 0x09, 0x60, 0x86, 0x48, 1, 101, 3, 4, 3

// These hash OIDs used in a lot of places.
#define OID_SHA1_VALUE 0x06, 0x05, 0x2B, 0x0E, 0x03, 0x02, 0x1A
SHA1_OID(_);  // Expands to:
              //     MAKE_OID(_SHA1)
              // which expands to:
              //     EXTERN  const BYTE OID_SHA1[] INITIALIZER({OID_SHA1_VALUE})
              // which, depending on the setting of EXTERN and
              // INITIALIZER, expands to either:
              //      extern const BYTE    OID_SHA1[]
              // or
              //      const BYTE           OID_SHA1[] = {OID_SHA1_VALUE}
              // which is:
              //      const BYTE           OID_SHA1[] = {0x06, 0x05, 0x2B, 0x0E,
              //                                         0x03, 0x02, 0x1A}

#define OID_SHA256_VALUE NIST_HASH, 1
SHA256_OID(_);

#define OID_SHA384_VALUE NIST_HASH, 2
SHA384_OID(_);

#define OID_SHA512_VALUE NIST_HASH, 3
SHA512_OID(_);

#define OID_SM3_256_VALUE 0x06, 0x08, 0x2A, 0x81, 0x1C, 0xCF, 0x55, 0x01, 0x83, 0x11
SM3_256_OID(_);  // (1.2.156.10197.1.401)

#define OID_SHA3_256_VALUE NIST_HASH, 8
SHA3_256_OID(_);

#define OID_SHA3_384_VALUE NIST_HASH, 9
SHA3_384_OID(_);

#define OID_SHA3_512_VALUE NIST_HASH, 10
SHA3_512_OID(_);

// These are used for RSA-PSS
#if ALG_RSA

#  define OID_MGF1_VALUE \
      0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x08
MAKE_OID(_MGF1);

#  define OID_RSAPSS_VALUE \
      0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x0A
MAKE_OID(_RSAPSS);

// This is the OID to designate the public part of an RSA key.
#  define OID_PKCS1_PUB_VALUE \
      0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01
MAKE_OID(_PKCS1_PUB);

// These are used for RSA PKCS1 signature Algorithms
#  define OID_PKCS1_SHA1_VALUE \
      0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x05
SHA1_OID(_PKCS1_);  // (1.2.840.113549.1.1.5)

#  define OID_PKCS1_SHA256_VALUE \
      0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x0B
SHA256_OID(_PKCS1_);  // (1.2.840.113549.1.1.11)
```

```
#  define OID_PKCS1_SHA384_VALUE \
      0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x0C
SHA384_OID(_PKCS1_);  // (1.2.840.113549.1.1.12)

#  define OID_PKCS1_SHA512_VALUE \
      0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x0D
SHA512_OID(_PKCS1_);  //(1.2.840.113549.1.1.13)

#  define OID_PKCS1_SM3_256_VALUE \
      0x06, 0x08, 0x2A, 0x81, 0x1C, 0xCF, 0x55, 0x01, 0x83, 0x78
SM3_256_OID(_PKCS1_);  // 1.2.156.10197.1.504

#  define OID_PKCS1_SHA3_256_VALUE NIST_SIG, 14
SHA3_256_OID(_PKCS1_);
#  define OID_PKCS1_SHA3_384_VALUE NIST_SIG, 15
SHA3_384_OID(_PKCS1_);
#  define OID_PKCS1_SHA3_512_VALUE NIST_SIG, 16
SHA3_512_OID(_PKCS1_);

#endif  // ALG_RSA

#if ALG_ECDSA

#  define OID_ECDSA_SHA1_VALUE 0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x04, 0x01
SHA1_OID(_ECDSA_);  // (1.2.840.10045.4.1) SHA1 digest signed by an ECDSA key.

#  define OID_ECDSA_SHA256_VALUE \
      0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x04, 0x03, 0x02
SHA256_OID(_ECDSA_);  // (1.2.840.10045.4.3.2) SHA256 digest signed by an ECDSA key.

#  define OID_ECDSA_SHA384_VALUE \
      0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x04, 0x03, 0x03
SHA384_OID(_ECDSA_);  // (1.2.840.10045.4.3.3) SHA384 digest signed by an ECDSA key.

#  define OID_ECDSA_SHA512_VALUE \
      0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x04, 0x03, 0x04
SHA512_OID(_ECDSA_);  // (1.2.840.10045.4.3.4) SHA512 digest signed by an ECDSA key.

#  define OID_ECDSA_SM3_256_VALUE \
      0x06, 0x08, 0x2A, 0x81, 0x1C, 0xCF, 0x55, 0x01, 0x83, 0x75
SM3_256_OID(_ECDSA_);  // 1.2.156.10197.1.501

#  define OID_ECDSA_SHA3_256_VALUE NIST_SIG, 10
SHA3_256_OID(_ECDSA_);
#  define OID_ECDSA_SHA3_384_VALUE NIST_SIG, 11
SHA3_384_OID(_ECDSA_);
#  define OID_ECDSA_SHA3_512_VALUE NIST_SIG, 12
SHA3_512_OID(_ECDSA_);

#endif  // ALG_ECDSA

#if ALG_ECC

#  define OID_ECC_PUBLIC_VALUE 0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x02, 0x01
MAKE_OID(_ECC_PUBLIC);

#  define OID_ECC_NIST_P192_VALUE \
      0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x03, 0x01, 0x01
#  if ECC_NIST_P192
MAKE_OID(_ECC_NIST_P192);  // (1.2.840.10045.3.1.1) 'nistP192'
#  endif                     // ECC_NIST_P192

#  define OID_ECC_NIST_P224_VALUE 0x06, 0x05, 0x2B, 0x81, 0x04, 0x00, 0x21
#  if ECC_NIST_P224
MAKE_OID(_ECC_NIST_P224);  // (1.3.132.0.33)        'nistP224'
```

```
#  endif                    // ECC_NIST_P224

#  define OID_ECC_NIST_P256_VALUE \
      0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x03, 0x01, 0x07
#  if ECC_NIST_P256
MAKE_OID(_ECC_NIST_P256);  // (1.2.840.10045.3.1.7)  'nistP256'
#  endif                   // ECC_NIST_P256

#  define OID_ECC_NIST_P384_VALUE 0x06, 0x05, 0x2B, 0x81, 0x04, 0x00, 0x22
#  if ECC_NIST_P384
MAKE_OID(_ECC_NIST_P384);  // (1.3.132.0.34)         'nistP384'
#  endif                   // ECC_NIST_P384

#  define OID_ECC_NIST_P521_VALUE 0x06, 0x05, 0x2B, 0x81, 0x04, 0x00, 0x23
#  if ECC_NIST_P521
MAKE_OID(_ECC_NIST_P521);  // (1.3.132.0.35)         'nistP521'
#  endif                   // ECC_NIST_P521

// No OIDs defined for these anonymous curves
#  define OID_ECC_BN_P256_VALUE 0x00
#  if ECC_BN_P256
MAKE_OID(_ECC_BN_P256);
#  endif  // ECC_BN_P256

#  define OID_ECC_BN_P638_VALUE 0x00
#  if ECC_BN_P638
MAKE_OID(_ECC_BN_P638);
#  endif  // ECC_BN_P638

#  define OID_ECC_SM2_P256_VALUE \
      0x06, 0x08, 0x2A, 0x81, 0x1C, 0xCF, 0x55, 0x01, 0x82, 0x2D
#  if ECC_SM2_P256
MAKE_OID(_ECC_SM2_P256);  // Don't know where I found this OID. It needs checking
#  endif                   // ECC_SM2_P256

#  if ECC_BN_P256
#    define OID_ECC_BN_P256 NULL
#  endif  // ECC_BN_P256

#endif  // ALG_ECC

#define OID_SIZE(OID) (OID[1] + 2)

#endif  // !_OIDS_H_
```

6.32 /tpm/include/private/PRNG_TestVectors.h

```
#ifndef _MSBN_DRBG_TEST_VECTORS_H
#define _MSBN_DRBG_TEST_VECTORS_H

//#if DRBG_ALGORITHM == TPM_ALG_AES && DRBG_KEY_BITS == 256
#if DRBG_KEY_SIZE_BITS == 256

/*(NIST test vector)
[AES-256 no df]
[PredictionResistance = False]
[EntropyInputLen = 384]
[NonceLen = 128]
[PersonalizationStringLen = 0]
[AdditionalInputLen = 0]

COUNT = 0
EntropyInput = 0d15aa80 b16c3a10 906cfedb 795dae0b 5b81041c 5c5bfacb
               373d4440 d9120f7e 3d6cf909 86cf52d8 5d3e947d 8c061f91
Nonce = 06caef5f b538e08e 1f3b0452 03f8f4b2
```

```
PersonalizationString =
AdditionalInput =
    INTERMEDIATE Key = be5df629 34cc1230 166a6773 345bbd6b
                       4c8869cf 8aec1c3b 1aa98bca 37cacf61
    INTERMEDIATE V = 3182dd1e 7638ec70 014e93bd 813e524c
    INTERMEDIATE ReturnedBits = 28e0ebb8 21016650 8c8f65f2 207bd0a3
EntropyInputReseed = 6ee793a3 3955d72a d12fd80a 8a3fcf95 ed3b4dac 5795fe25
                     cf869f7c 27573bbc 56f1acae 13a65042 b340093c 464a7a22
AdditionalInputReseed =
AdditionalInput =
ReturnedBits = 946f5182 d54510b9 461248f5 71ca06c9
*/

// Entropy is the size of the state. The state is the size of the key
// plus the IV. The IV is a block. If Key = 256 and Block = 128 then State = 384
#  define DRBG_TEST_INITIATE_ENTROPY                                          \
      0x0d, 0x15, 0xaa, 0x80, 0xb1, 0x6c, 0x3a, 0x10, 0x90, 0x6c, 0xfe, 0xdb, 0x79, \
          0x5d, 0xae, 0x0b, 0x5b, 0x81, 0x04, 0x1c, 0x5c, 0x5b, 0xfa, 0xcb, 0x37,   \
          0x3d, 0x44, 0x40, 0xd9, 0x12, 0x0f, 0x7e, 0x3d, 0x6c, 0xf9, 0x09, 0x86,   \
          0xcf, 0x52, 0xd8, 0x5d, 0x3e, 0x94, 0x7d, 0x8c, 0x06, 0x1f, 0x91

#  define DRBG_TEST_RESEED_ENTROPY                                            \
      0x6e, 0xe7, 0x93, 0xa3, 0x39, 0x55, 0xd7, 0x2a, 0xd1, 0x2f, 0xd8, 0x0a, 0x8a, \
          0x3f, 0xcf, 0x95, 0xed, 0x3b, 0x4d, 0xac, 0x57, 0x95, 0xfe, 0x25, 0xcf,   \
          0x86, 0x9f, 0x7c, 0x27, 0x57, 0x3b, 0xbc, 0x56, 0xf1, 0xac, 0xae, 0x13,   \
          0xa6, 0x50, 0x42, 0xb3, 0x40, 0x09, 0x3c, 0x46, 0x4a, 0x7a, 0x22

#  define DRBG_TEST_GENERATED_INTERM                                          \
      0x28, 0xe0, 0xeb, 0xb8, 0x21, 0x01, 0x66, 0x50, 0x8c, 0x8f, 0x65, 0xf2, 0x20, \
          0x7b, 0xd0, 0xa3

#  define DRBG_TEST_GENERATED                                                 \
      0x94, 0x6f, 0x51, 0x82, 0xd5, 0x45, 0x10, 0xb9, 0x46, 0x12, 0x48, 0xf5, 0x71, \
          0xca, 0x06, 0xc9
#elif DRBG_KEY_SIZE_BITS == 128
/*(NIST test vector)
[AES-128 no df]
[PredictionResistance = False]
[EntropyInputLen = 256]
[NonceLen = 64]
[PersonalizationStringLen = 0]
[AdditionalInputLen = 0]

COUNT = 0
EntropyInput = 8fc11bdb5aabb7e093b61428e0907303cb459f3b600dad870955f22da80a44f8
Nonce = be1f73885ddd15aa
PersonalizationString =
AdditionalInput =
    INTERMEDIATE Key = b134ecc836df6dbd624900af118dd7e6
    INTERMEDIATE V = 01bb09e86dabd75c9f26dbf6f9531368
    INTERMEDIATE ReturnedBits = dc3cf6bf5bd341135f2c6811a1071c87
EntropyInputReseed =
                0cd53cd5eccd5a10d7ea266111259b05574fc6ddd8bed8bd72378cf82f1dba2a
AdditionalInputReseed =
AdditionalInput =
ReturnedBits = b61850decfd7106d44769a8e6e8c1ad4
*/

#  define DRBG_TEST_INITIATE_ENTROPY                                          \
      0x8f, 0xc1, 0x1b, 0xdb, 0x5a, 0xab, 0xb7, 0xe0, 0x93, 0xb6, 0x14, 0x28, 0xe0, \
          0x90, 0x73, 0x03, 0xcb, 0x45, 0x9f, 0x3b, 0x60, 0x0d, 0xad, 0x87, 0x09,   \
          0x55, 0xf2, 0x2d, 0xa8, 0x0a, 0x44, 0xf8

#  define DRBG_TEST_RESEED_ENTROPY                                            \
      0x0c, 0xd5, 0x3c, 0xd5, 0xec, 0xcd, 0x5a, 0x10, 0xd7, 0xea, 0x26, 0x61, 0x11, \
          0x25, 0x9b, 0x05, 0x57, 0x4f, 0xc6, 0xdd, 0xd8, 0xbe, 0xd8, 0xbd, 0x72,   \
```

```
          0x37, 0x8c, 0xf8, 0x2f, 0x1d, 0xba, 0x2a

#  define DRBG_TEST_GENERATED_INTERM                                          \
      0xdc, 0x3c, 0xf6, 0xbf, 0x5b, 0xd3, 0x41, 0x13, 0x5f, 0x2c, 0x68, 0x11, 0xa1, \
          0x07, 0x1c, 0x87

#  define DRBG_TEST_GENERATED                                                 \
      0xb6, 0x18, 0x50, 0xde, 0xcf, 0xd7, 0x10, 0x6d, 0x44, 0x76, 0x9a, 0x8e, 0x6e, \
          0x8c, 0x1a, 0xd4

#endif

#endif  //     _MSBN_DRBG_TEST_VECTORS_H
```

6.33 /tpm/include/private/RsaTestData.h

```
//
// RSA Test Vectors

#define RSA_TEST_KEY_SIZE 256

typedef struct
{
    UINT16 size;
    BYTE   buffer[RSA_TEST_KEY_SIZE];
} TPM2B_RSA_TEST_KEY;

typedef TPM2B_RSA_TEST_KEY TPM2B_RSA_TEST_VALUE;

typedef struct
{
    UINT16 size;
    BYTE   buffer[RSA_TEST_KEY_SIZE / 2];
} TPM2B_RSA_TEST_PRIME;

const TPM2B_RSA_TEST_KEY c_rsaPublicModulus =
    {256,
     {0x91, 0x12, 0xf5, 0x07, 0x9d, 0x5f, 0x6b, 0x1c, 0x90, 0xf6, 0xcc, 0x87, 0xde,
      0x3a, 0x7a, 0x15, 0xdc, 0x54, 0x07, 0x6c, 0x26, 0x8f, 0x25, 0xef, 0x7e, 0x66,
      0xc0, 0xe3, 0x82, 0x12, 0x2f, 0xab, 0x52, 0x82, 0x1e, 0x85, 0xbc, 0x53, 0xba,
      0x2b, 0x01, 0xad, 0x01, 0xc7, 0x8d, 0x46, 0x4f, 0x7d, 0xdd, 0x7e, 0xdc, 0xb0,
      0xad, 0xf6, 0x0c, 0xa1, 0x62, 0x92, 0x97, 0x8a, 0x3e, 0x6f, 0x7e, 0x3e, 0xf6,
      0x9a, 0xcc, 0xf9, 0xa9, 0x86, 0x77, 0xb6, 0x85, 0x43, 0x42, 0x04, 0x13, 0x65,
      0xe2, 0xad, 0x36, 0xc9, 0xbf, 0xc1, 0x97, 0x84, 0x6f, 0xee, 0x7c, 0xda, 0x58,
      0xd2, 0xae, 0x07, 0x00, 0xaf, 0xc5, 0x5f, 0x4d, 0x3a, 0x98, 0xb0, 0xed, 0x27,
      0x7c, 0xc2, 0xce, 0x26, 0x5d, 0x87, 0xe1, 0xe3, 0xa9, 0x69, 0x88, 0x4f, 0x8c,
      0x08, 0x31, 0x18, 0xae, 0x93, 0x16, 0xe3, 0x74, 0xde, 0xd3, 0xf6, 0x16, 0xaf,
      0xa3, 0xac, 0x37, 0x91, 0x8d, 0x10, 0xc6, 0x6b, 0x64, 0x14, 0x3a, 0xd9, 0xfc,
      0xe4, 0xa0, 0xf2, 0xd1, 0x01, 0x37, 0x4f, 0x4a, 0xeb, 0xe5, 0xec, 0x98, 0xc5,
      0xd9, 0x4b, 0x30, 0xd2, 0x80, 0x2a, 0x5a, 0x18, 0x5a, 0x7d, 0xd4, 0x3d, 0xb7,
      0x62, 0x98, 0xce, 0x6d, 0xa2, 0x02, 0x6e, 0x45, 0xaa, 0x95, 0x73, 0xe0, 0xaa,
      0x75, 0x57, 0xb1, 0x3d, 0x1b, 0x05, 0x75, 0x23, 0x6b, 0x20, 0x69, 0x9e, 0x14,
      0xb0, 0x7f, 0xac, 0xae, 0xd2, 0xc7, 0x48, 0x3b, 0xe4, 0x56, 0x11, 0x34, 0x1e,
      0x05, 0x1a, 0x30, 0x20, 0xef, 0x68, 0x93, 0x6b, 0x9d, 0x7e, 0xdd, 0xba, 0x96,
      0x50, 0xcc, 0x1c, 0x81, 0xb4, 0x59, 0xb9, 0x74, 0x36, 0xd9, 0x97, 0xdc, 0x8f,
      0x17, 0x82, 0x72, 0xb3, 0x59, 0xf6, 0x23, 0xfa, 0x84, 0xf7, 0x6d, 0xf2, 0x05,
      0xff, 0xf1, 0xb9, 0xcc, 0xe9, 0xa2, 0x82, 0x01, 0xfb}};

const TPM2B_RSA_TEST_PRIME c_rsaPrivatePrime =
    {RSA_TEST_KEY_SIZE / 2,
     {0xb7, 0xa0, 0x90, 0xc7, 0x92, 0x09, 0xde, 0x71, 0x03, 0x37, 0x4a, 0xb5, 0x2f,
      0xda, 0x61, 0xb8, 0x09, 0x1b, 0xba, 0x99, 0x70, 0x45, 0xc1, 0x0b, 0x15, 0x12,
      0x71, 0x8a, 0xb3, 0x2a, 0x4d, 0x5a, 0x41, 0x9b, 0x73, 0x89, 0x80, 0x0a, 0x8f,
      0x18, 0x4c, 0x8b, 0xa2, 0x5b, 0xda, 0xbd, 0x43, 0xbe, 0xdc, 0x76, 0x4d, 0x71,
      0x0f, 0xb9, 0xfc, 0x7a, 0x09, 0xfe, 0x4f, 0xac, 0x63, 0xd9, 0x2e, 0x50, 0x3a,
```

```
      0xa1, 0x37, 0xc6, 0xf2, 0xa1, 0x89, 0x12, 0xe7, 0x72, 0x64, 0x2b, 0xba, 0xc1,
      0x1f, 0xca, 0x9d, 0xb7, 0xaa, 0x3a, 0xa9, 0xd3, 0xa6, 0x6f, 0x73, 0x02, 0xbb,
      0x85, 0x5d, 0x9a, 0xb9, 0x5c, 0x08, 0x83, 0x22, 0x20, 0x49, 0x91, 0x5f, 0x4b,
      0x86, 0xbc, 0x3f, 0x76, 0x43, 0x08, 0x97, 0xbf, 0x82, 0x55, 0x36, 0x2d, 0x8b,
      0x6e, 0x9e, 0xfb, 0xc1, 0x67, 0x6a, 0x43, 0xa2, 0x46, 0x81, 0x71}};

const BYTE c_RsaTestValue[RSA_TEST_KEY_SIZE] =
    {0x2a, 0x24, 0x3a, 0xbb, 0x50, 0x1d, 0xd4, 0x2a, 0xf9, 0x18, 0x32, 0x34, 0xa2,
     0x0f, 0xea, 0x5c, 0x91, 0x77, 0xe9, 0xe1, 0x09, 0x83, 0xdc, 0x5f, 0x71, 0x64,
     0x5b, 0xeb, 0x57, 0x79, 0xa0, 0x41, 0xc9, 0xe4, 0x5a, 0x0b, 0xf4, 0x9f, 0xdb,
     0x84, 0x04, 0xa6, 0x48, 0x24, 0xf6, 0x3f, 0x66, 0x1f, 0xa8, 0x04, 0x5c, 0xf0,
     0x7a, 0x6b, 0x4a, 0x9c, 0x7e, 0x21, 0xb6, 0xda, 0x6b, 0x65, 0x9c, 0x3a, 0x68,
     0x50, 0x13, 0x1e, 0xa4, 0xb7, 0xca, 0xec, 0xd3, 0xcc, 0xb2, 0x9b, 0x8c, 0x87,
     0xa4, 0x6a, 0xba, 0xc2, 0x06, 0x3f, 0x40, 0x48, 0x7b, 0xa8, 0xb8, 0x2c, 0x03,
     0x14, 0x33, 0xf3, 0x1d, 0xe9, 0xbd, 0x6f, 0x54, 0x66, 0xb4, 0x69, 0x5e, 0xbc,
     0x80, 0x7c, 0xe9, 0x6a, 0x43, 0x7f, 0xb8, 0x6a, 0xa0, 0x5f, 0x5d, 0x7a, 0x20,
     0xfd, 0x7a, 0x39, 0xe1, 0xea, 0x0e, 0x94, 0x91, 0x28, 0x63, 0x7a, 0xac, 0xc9,
     0xa5, 0x3a, 0x6d, 0x31, 0x7b, 0x7c, 0x54, 0x56, 0x99, 0x56, 0xbb, 0xb7, 0xa1,
     0x2d, 0xd2, 0x5c, 0x91, 0x5f, 0x1c, 0xd3, 0x06, 0x7f, 0x34, 0x53, 0x2f, 0x4c,
     0xd1, 0x8b, 0xd2, 0x9e, 0xdc, 0xc3, 0x94, 0x0a, 0xe1, 0x0f, 0xa5, 0x15, 0x46,
     0x2a, 0x8e, 0x10, 0xc2, 0xfe, 0xb7, 0x5e, 0x2d, 0x0d, 0xd1, 0x25, 0xfc, 0xe4,
     0xf7, 0x02, 0x19, 0xfe, 0xb6, 0xe4, 0x95, 0x9c, 0x17, 0x4a, 0x9b, 0xdb, 0xab,
     0xc7, 0x79, 0xe3, 0x5e, 0x40, 0xd0, 0x56, 0x6d, 0x25, 0x0a, 0x72, 0x65, 0x80,
     0x92, 0x9a, 0xa8, 0x07, 0x70, 0x32, 0x14, 0xfb, 0xfe, 0x08, 0xeb, 0x13, 0xb4,
     0x07, 0x68, 0xb4, 0x58, 0x39, 0xbe, 0x8e, 0x78, 0x3a, 0x59, 0x3f, 0x9c, 0x4c,
     0xe9, 0xa8, 0x64, 0x68, 0xf7, 0xb9, 0x6e, 0x20, 0xf5, 0xcb, 0xca, 0x47, 0xf2,
     0x17, 0xaa, 0x8b, 0xbc, 0x13, 0x14, 0x84, 0xf6, 0xab};

const TPM2B_RSA_TEST_VALUE c_RsaepKvt =
    {RSA_TEST_KEY_SIZE,
     {0x73, 0xbd, 0x65, 0x49, 0xda, 0x7b, 0xb8, 0x50, 0x9e, 0x87, 0xf0, 0x0a, 0x8a,
      0x9a, 0x07, 0xb6, 0x00, 0x82, 0x10, 0x14, 0x60, 0xd8, 0x01, 0xfc, 0xc5, 0x18,
      0xea, 0x49, 0x5f, 0x13, 0xcf, 0x65, 0x66, 0x30, 0x6c, 0x60, 0x3f, 0x24, 0x3c,
      0xfb, 0xe2, 0x31, 0x16, 0x99, 0x7e, 0x31, 0x98, 0xab, 0x93, 0xb8, 0x07, 0x53,
      0xcc, 0xdb, 0x7f, 0x44, 0xd9, 0xee, 0x5d, 0xe8, 0x5f, 0x97, 0x5f, 0xe8, 0x1f,
      0x88, 0x52, 0x24, 0x7b, 0xac, 0x62, 0x95, 0xb7, 0x7d, 0xf5, 0xf8, 0x9f, 0x5a,
      0xa8, 0x24, 0x9a, 0x76, 0x71, 0x2a, 0x35, 0x2a, 0xa1, 0x08, 0xbb, 0x95, 0xe3,
      0x64, 0xdc, 0xdb, 0xc2, 0x33, 0xa9, 0x5f, 0xbe, 0x4c, 0xc4, 0xcc, 0x28, 0xc9,
      0x25, 0xff, 0xee, 0x17, 0x15, 0x9a, 0x50, 0x90, 0x0e, 0x15, 0xb4, 0xea, 0x6a,
      0x09, 0xe6, 0xff, 0xa4, 0xee, 0xc7, 0x7e, 0xce, 0xa9, 0x73, 0xe4, 0xa0, 0x56,
      0xbd, 0x53, 0x2a, 0xe4, 0xc0, 0x2b, 0xa8, 0x9b, 0x09, 0x30, 0x72, 0x62, 0x0f,
      0xf9, 0xf6, 0xa1, 0x52, 0xd2, 0x8a, 0x37, 0xee, 0xa5, 0xc8, 0x47, 0xe1, 0x99,
      0x21, 0x47, 0xeb, 0xdd, 0x37, 0xaa, 0xe4, 0xbd, 0x55, 0x46, 0x5a, 0x5a, 0x5d,
      0xfb, 0x7b, 0xfc, 0xff, 0xbf, 0x26, 0x71, 0xf6, 0x1e, 0xad, 0xbc, 0xbf, 0x33,
      0xca, 0xe1, 0x92, 0x8f, 0x2a, 0x89, 0x6c, 0x45, 0x24, 0xd1, 0xa6, 0x52, 0x56,
      0x24, 0x5e, 0x90, 0x47, 0xe5, 0xcb, 0x12, 0xb0, 0x32, 0xf9, 0xa6, 0xbb, 0xea,
      0x37, 0xa9, 0xbd, 0xef, 0x23, 0xef, 0x63, 0x07, 0x6c, 0xc4, 0x4e, 0x64, 0x3c,
      0xc6, 0x11, 0x84, 0x7d, 0x65, 0xd6, 0x5d, 0x7a, 0x17, 0x58, 0xa5, 0xf7, 0x74,
      0x3b, 0x42, 0xe3, 0xd2, 0xda, 0x5f, 0x6f, 0xe0, 0x1e, 0x4b, 0xcf, 0x46, 0xe2,
      0xdf, 0x3e, 0x41, 0x8e, 0x0e, 0xb0, 0x3f, 0x8b, 0x65}};

#define OAEP_TEST_LABEL "OAEP Test Value"

#if ALG_SHA1_VALUE == DEFAULT_TEST_HASH

const TPM2B_RSA_TEST_VALUE c_OaepKvt =
    {RSA_TEST_KEY_SIZE,
     {0x32, 0x68, 0x84, 0x0b, 0x9c, 0xc9, 0x25, 0x26, 0xd9, 0xc0, 0xd0, 0xb1, 0xde,
      0x60, 0x55, 0xae, 0x33, 0xe5, 0xcf, 0x6c, 0x85, 0xbe, 0x0d, 0x71, 0x11, 0xe1,
      0x45, 0x60, 0xbb, 0x42, 0x3d, 0xf3, 0xb1, 0x18, 0x84, 0x7b, 0xc6, 0x5d, 0xce,
      0x1d, 0x5f, 0x9a, 0x97, 0xcf, 0xb1, 0x97, 0x9a, 0x85, 0x7c, 0xa7, 0xa1, 0x63,
      0x23, 0xb6, 0x74, 0x0f, 0x1a, 0xee, 0x29, 0x51, 0xeb, 0x50, 0x8f, 0x3c, 0x8e,
      0x4e, 0x31, 0x38, 0xdc, 0x11, 0xfc, 0x9a, 0x4e, 0xaf, 0x93, 0xc9, 0x7f, 0x6e,
      0x35, 0xf3, 0xc9, 0xe4, 0x89, 0x14, 0x53, 0xe2, 0xc2, 0x1a, 0xf7, 0x6b, 0x9b,
      0xf0, 0x7a, 0xa4, 0x69, 0x52, 0xe0, 0x24, 0x8f, 0xea, 0x31, 0xa7, 0x5c, 0x43,
      0xb0, 0x65, 0xc9, 0xfe, 0xba, 0xfe, 0x80, 0x9e, 0xa5, 0xc0, 0xf5, 0x8d, 0xce,
```

```
      0x41, 0xf9, 0x83, 0x0d, 0x8e, 0x0f, 0xef, 0x3d, 0x1f, 0x6a, 0xcc, 0x8a, 0x3d,
      0x3b, 0xdf, 0x22, 0x38, 0xd7, 0x34, 0x58, 0x7b, 0x55, 0xc9, 0xf6, 0xbc, 0x7c,
      0x4c, 0x3f, 0xd7, 0xde, 0x4e, 0x30, 0xa9, 0x69, 0xf3, 0x5f, 0x56, 0x8f, 0xc2,
      0xe7, 0x75, 0x79, 0xb8, 0xa5, 0xc8, 0x0d, 0xc0, 0xcd, 0xb6, 0xc9, 0x63, 0xad,
      0x7c, 0xe4, 0x8f, 0x39, 0x60, 0x4d, 0x7d, 0xdb, 0x34, 0x49, 0x2a, 0x47, 0xde,
      0xc0, 0x42, 0x4a, 0x19, 0x94, 0x2e, 0x50, 0x21, 0x03, 0x47, 0xff, 0x73, 0xb3,
      0xb7, 0x89, 0xcc, 0x7b, 0x2c, 0xeb, 0x03, 0xa7, 0x9a, 0x06, 0xfd, 0xed, 0x19,
      0xbb, 0x82, 0xa0, 0x13, 0xe9, 0xfa, 0xac, 0x06, 0x5f, 0xc5, 0xa9, 0x2b, 0xda,
      0x88, 0x23, 0xa2, 0x5d, 0xc2, 0x7f, 0xda, 0xc8, 0x5a, 0x94, 0x31, 0xc1, 0x21,
      0xd7, 0x1e, 0x6b, 0xd7, 0x89, 0xb1, 0x93, 0x80, 0xab, 0xd1, 0x37, 0xf2, 0x6f,
      0x50, 0xcd, 0x2a, 0xea, 0xb1, 0xc4, 0xcd, 0xcb, 0xb5}};

const TPM2B_RSA_TEST_VALUE c_RsaesKvt =
    {RSA_TEST_KEY_SIZE,
     {0x29, 0xa4, 0x2f, 0xbb, 0x8a, 0x14, 0x05, 0x1e, 0x3c, 0x72, 0x76, 0x77, 0x38,
      0xe7, 0x73, 0xe3, 0x6e, 0x24, 0x4b, 0x38, 0xd2, 0x1a, 0xcf, 0x23, 0x58, 0x78,
      0x36, 0x82, 0x23, 0x6e, 0x6b, 0xef, 0x2c, 0x3d, 0xf2, 0xe8, 0xd6, 0xc6, 0x87,
      0x8e, 0x78, 0x9b, 0x27, 0x39, 0xc0, 0xd6, 0xef, 0x4d, 0x0b, 0xfc, 0x51, 0x27,
      0x18, 0xf3, 0x51, 0x5e, 0x4d, 0x96, 0x3a, 0xe2, 0x15, 0xe2, 0x7e, 0x42, 0xf4,
      0x16, 0xd5, 0xc6, 0x52, 0x5d, 0x17, 0x44, 0x76, 0x09, 0x7a, 0xcf, 0xe3, 0x30,
      0xe3, 0x84, 0xf6, 0x6f, 0x3a, 0x33, 0xfb, 0x32, 0x0d, 0x1d, 0xe7, 0x7c, 0x80,
      0x82, 0x4f, 0xed, 0xda, 0x87, 0x11, 0x9c, 0xc3, 0x7e, 0x85, 0xbd, 0x18, 0x58,
      0x08, 0x2b, 0x23, 0x37, 0xe7, 0x9d, 0xd0, 0xd1, 0x79, 0xe2, 0x05, 0xbd, 0xf5,
      0x4f, 0x0e, 0x0f, 0xdb, 0x4a, 0x74, 0xeb, 0x09, 0x01, 0xb3, 0xca, 0xbd, 0xa6,
      0x7b, 0x09, 0xb1, 0x13, 0x77, 0x30, 0x4d, 0x87, 0x41, 0x06, 0x57, 0x2e, 0x5f,
      0x36, 0x6e, 0xfc, 0x35, 0x69, 0xfe, 0x0a, 0x24, 0x6c, 0x98, 0x8c, 0xda, 0x97,
      0xf4, 0xfb, 0xc7, 0x83, 0x2d, 0x3e, 0x7d, 0xc0, 0x5c, 0x34, 0xfd, 0x11, 0x2a,
      0x12, 0xa7, 0xae, 0x4a, 0xde, 0xc8, 0x4e, 0xcf, 0xf4, 0x85, 0x63, 0x77, 0xc6,
      0x33, 0x34, 0xe0, 0x27, 0xe4, 0x9e, 0x91, 0x0b, 0x4b, 0x85, 0xf0, 0xb0, 0x79,
      0xaa, 0x7c, 0xc6, 0xff, 0x3b, 0xbc, 0x04, 0x73, 0xb8, 0x95, 0xd7, 0x31, 0x54,
      0x3b, 0x56, 0xec, 0x52, 0x15, 0xd7, 0x3e, 0x62, 0xf5, 0x82, 0x99, 0x3e, 0x2a,
      0xc0, 0x4b, 0x2e, 0x06, 0x57, 0x6d, 0x3f, 0x3e, 0x77, 0x1f, 0x2b, 0x2d, 0xc5,
      0xb9, 0x3b, 0x68, 0x56, 0x73, 0x70, 0x32, 0x6b, 0x6b, 0x65, 0x25, 0x76, 0x45,
      0x6c, 0x45, 0xf1, 0x6c, 0x59, 0xfc, 0x94, 0xa7, 0x15}};

const TPM2B_RSA_TEST_VALUE c_RsapssKvt =
    {RSA_TEST_KEY_SIZE,
     {0x01, 0xfe, 0xd5, 0x83, 0x0b, 0x15, 0xba, 0x90, 0x2c, 0xdf, 0xf7, 0x26, 0xb7,
      0x8f, 0xb1, 0xd7, 0x0b, 0xfd, 0x83, 0xf9, 0x95, 0xd5, 0xd7, 0xb5, 0xc5, 0xc5,
      0x4a, 0xde, 0xd5, 0xe6, 0x20, 0x78, 0xca, 0x73, 0x77, 0x3d, 0x61, 0x36, 0x48,
      0xae, 0x3e, 0x8f, 0xee, 0x43, 0x29, 0x96, 0xdf, 0x3f, 0x1c, 0x97, 0x5a, 0xbe,
      0xe5, 0xa2, 0x7e, 0x5b, 0xd0, 0xc0, 0x29, 0x39, 0x83, 0x81, 0x77, 0x24, 0x43,
      0xdb, 0x3c, 0x64, 0x4d, 0xf0, 0x23, 0xe4, 0xae, 0x0f, 0x78, 0x31, 0x8c, 0xda,
      0x0c, 0xec, 0xf1, 0xdf, 0x09, 0xf2, 0x14, 0x6a, 0x4d, 0xaf, 0x36, 0x81, 0x6e,
      0xbd, 0xbe, 0x36, 0x79, 0x88, 0x98, 0xb6, 0x6f, 0x5a, 0xad, 0xcf, 0x7c, 0xee,
      0xe0, 0xdd, 0x00, 0xbe, 0x59, 0x97, 0x88, 0x00, 0x34, 0xc0, 0x8b, 0x48, 0x42,
      0x05, 0x04, 0x5a, 0xb7, 0x85, 0x38, 0xa0, 0x35, 0xd7, 0x3b, 0x51, 0xb8, 0x7b,
      0x81, 0x83, 0xee, 0xff, 0x76, 0x6f, 0x50, 0x39, 0x4d, 0xab, 0x89, 0x63, 0x07,
      0x6d, 0xf5, 0xe5, 0x01, 0x10, 0x56, 0xfe, 0x93, 0x06, 0x8f, 0xd3, 0xc9, 0x41,
      0xab, 0xc9, 0xdf, 0x6e, 0x59, 0xa8, 0xc3, 0x1d, 0xbf, 0x96, 0x4a, 0x59, 0x80,
      0x3c, 0x90, 0x3a, 0x59, 0x56, 0x4c, 0x6d, 0x44, 0x6d, 0xeb, 0xdc, 0x73, 0xcd,
      0xc1, 0xec, 0xb8, 0x41, 0xbf, 0x89, 0x8c, 0x03, 0x69, 0x4c, 0xaf, 0x3f, 0xc1,
      0xc5, 0xc7, 0xe7, 0x7d, 0xa7, 0x83, 0x39, 0x70, 0xa2, 0x6b, 0x83, 0xbc, 0xbe,
      0xf5, 0xbf, 0x1c, 0xee, 0x6e, 0xa3, 0x22, 0x1e, 0x25, 0x2f, 0x16, 0x68, 0x69,
      0x5a, 0x1d, 0xfa, 0x2c, 0x3a, 0x0f, 0x67, 0xe1, 0x77, 0x12, 0xe8, 0x3d, 0xba,
      0xaa, 0xef, 0x96, 0x9c, 0x1f, 0x64, 0x32, 0xf4, 0xa7, 0xb3, 0x3f, 0x7d, 0x61,
      0xbb, 0x9a, 0x27, 0xad, 0xfb, 0x2f, 0x33, 0xc4, 0x70}};

const TPM2B_RSA_TEST_VALUE c_RsassaKvt =
    {RSA_TEST_KEY_SIZE,
     {0x67, 0x4e, 0xdd, 0xc2, 0xd2, 0x6d, 0xe0, 0x03, 0xc4, 0xc2, 0x41, 0xd3, 0xd4,
      0x61, 0x30, 0xd0, 0xe1, 0x68, 0x31, 0x4a, 0xda, 0xd9, 0xc2, 0x5d, 0xaa, 0xa2,
      0x7b, 0xfb, 0x44, 0x02, 0xf5, 0xd6, 0xd8, 0x2e, 0xcd, 0x13, 0x36, 0xc9, 0x4b,
      0xdb, 0x1a, 0x4b, 0x66, 0x1b, 0x4f, 0x9c, 0xb7, 0x17, 0xac, 0x53, 0x37, 0x4f,
      0x21, 0xbd, 0x0c, 0x66, 0xac, 0x06, 0x65, 0x52, 0x9f, 0x04, 0xf6, 0xa5, 0x22,
      0x5b, 0xf7, 0xe6, 0x0d, 0x3c, 0x9f, 0x41, 0x19, 0x09, 0x88, 0x7c, 0x41, 0x4c,
```

```
      0x2f, 0x9c, 0x8b, 0x3c, 0xdd, 0x7c, 0x28, 0x78, 0x24, 0xd2, 0x09, 0xa6, 0x5b,
      0xf7, 0x3c, 0x88, 0x7e, 0x73, 0x5a, 0x2d, 0x36, 0x02, 0x4f, 0x65, 0xb0, 0xcb,
      0xc8, 0xdc, 0xac, 0xa2, 0xda, 0x8b, 0x84, 0x91, 0x71, 0xe4, 0x30, 0x8b, 0xb6,
      0x12, 0xf2, 0xf0, 0xd0, 0xa0, 0x38, 0xcf, 0x75, 0xb7, 0x20, 0xcb, 0x35, 0x51,
      0x52, 0x6b, 0xc4, 0xf4, 0x21, 0x95, 0xc2, 0xf7, 0x9a, 0x13, 0xc1, 0x1a, 0x7b,
      0x8f, 0x77, 0xda, 0x19, 0x48, 0xbb, 0x6d, 0x14, 0x5d, 0xba, 0x65, 0xb4, 0x9e,
      0x43, 0x42, 0x58, 0x98, 0x0b, 0x91, 0x46, 0xd8, 0x4c, 0xf3, 0x4c, 0xaf, 0x2e,
      0x02, 0xa6, 0xb2, 0x49, 0x12, 0x62, 0x43, 0x4e, 0xa8, 0xac, 0xbf, 0xfd, 0xfa,
      0x37, 0x24, 0xea, 0x69, 0x1c, 0xf5, 0xae, 0xfa, 0x08, 0x82, 0x30, 0xc3, 0xc0,
      0xf8, 0x9a, 0x89, 0x33, 0xe1, 0x40, 0x6d, 0x18, 0x5c, 0x7b, 0x90, 0x48, 0xbf,
      0x37, 0xdb, 0xea, 0xfb, 0x0e, 0xd4, 0x2e, 0x11, 0xfa, 0xa9, 0x86, 0xff, 0x00,
      0x0b, 0x7b, 0xca, 0x09, 0x64, 0x6a, 0x8f, 0x0c, 0x0e, 0x09, 0x14, 0x36, 0x4a,
      0x74, 0x31, 0x18, 0x5b, 0x18, 0xeb, 0xea, 0x83, 0xc3, 0x66, 0x68, 0xa6, 0x7d,
      0x43, 0x06, 0x0f, 0x99, 0x60, 0xce, 0x65, 0x08, 0xf6}};

#endif  // SHA1

#if ALG_SHA256_VALUE == DEFAULT_TEST_HASH

const TPM2B_RSA_TEST_VALUE c_OaepKvt =
    {RSA_TEST_KEY_SIZE,
     {0x33, 0x20, 0x6e, 0x21, 0xc3, 0xf6, 0xcd, 0xf8, 0xd7, 0x5d, 0x9f, 0xe9, 0x05,
      0x14, 0x8c, 0x7c, 0xbb, 0x69, 0x24, 0x9e, 0x52, 0x8f, 0xaf, 0x84, 0x73, 0x21,
      0x2c, 0x85, 0xa5, 0x30, 0x4d, 0xb6, 0xb8, 0xfa, 0x15, 0x9b, 0xc7, 0x8f, 0xc9,
      0x7a, 0x72, 0x4b, 0x85, 0xa4, 0x1c, 0xc5, 0xd8, 0xe4, 0x92, 0xb3, 0xec, 0xd9,
      0xa8, 0xca, 0x5e, 0x74, 0x73, 0x89, 0x7f, 0xb4, 0xac, 0x7e, 0x68, 0x12, 0xb2,
      0x53, 0x27, 0x4b, 0xbf, 0xd0, 0x71, 0x69, 0x46, 0x9f, 0xef, 0xf4, 0x70, 0x60,
      0xf8, 0xd7, 0xae, 0xc7, 0x5a, 0x27, 0x38, 0x25, 0x2d, 0x25, 0xab, 0x96, 0x56,
      0x66, 0x3a, 0x23, 0x40, 0xa8, 0xdb, 0xbc, 0x86, 0xe8, 0xf3, 0xd2, 0x58, 0x0b,
      0x44, 0xfc, 0x94, 0x1e, 0xb7, 0x5d, 0xb4, 0x57, 0xb5, 0xf3, 0x56, 0xee, 0x9b,
      0xcf, 0x97, 0x91, 0x29, 0x36, 0xe3, 0x06, 0x13, 0xa2, 0xea, 0xd6, 0xd6, 0x0b,
      0x86, 0x0b, 0x1a, 0x27, 0xe6, 0x22, 0xc4, 0x7b, 0xff, 0xde, 0x0f, 0xbf, 0x79,
      0xc8, 0x1b, 0xed, 0xf1, 0x27, 0x62, 0xb5, 0x8b, 0xf9, 0xd9, 0x76, 0x90, 0xf6,
      0xcc, 0x83, 0x0f, 0xce, 0xce, 0x2e, 0x63, 0x7a, 0x9b, 0xf4, 0x48, 0x5b, 0xd7,
      0x81, 0x2c, 0x3a, 0xdb, 0x59, 0x0d, 0x4d, 0x9e, 0x46, 0xe9, 0x9e, 0x92, 0x22,
      0x27, 0x1c, 0xb0, 0x67, 0x8a, 0xe6, 0x8a, 0x16, 0x8a, 0xdf, 0x95, 0x76, 0x24,
      0x82, 0xad, 0xf1, 0xbc, 0x97, 0xbf, 0xd3, 0x5e, 0x6e, 0x14, 0x0c, 0x5b, 0x25,
      0xfe, 0x58, 0xfa, 0x64, 0xe5, 0x14, 0x46, 0xb7, 0x58, 0xc6, 0x3f, 0x7f, 0x42,
      0xd2, 0x8e, 0x45, 0x13, 0x41, 0x85, 0x12, 0x2e, 0x96, 0x19, 0xd0, 0x5e, 0x7d,
      0x34, 0x06, 0x32, 0x2b, 0xc8, 0xd9, 0x0d, 0x6c, 0x06, 0x36, 0xa0, 0xff, 0x47,
      0x57, 0x2c, 0x25, 0xbc, 0x8a, 0xa5, 0xe2, 0xc7, 0xe3}};

const TPM2B_RSA_TEST_VALUE c_RsaesKvt =
    {RSA_TEST_KEY_SIZE,
     {0x39, 0xfc, 0x10, 0x5d, 0xf4, 0x45, 0x3d, 0x94, 0x53, 0x06, 0x89, 0x24, 0xe7,
      0xe8, 0xfd, 0x03, 0xac, 0xfd, 0xbd, 0xb2, 0x28, 0xd3, 0x4a, 0x52, 0xc5, 0xd4,
      0xdb, 0x17, 0xd4, 0x24, 0x05, 0xc4, 0xeb, 0x6a, 0xce, 0x1d, 0xbb, 0x37, 0xcb,
      0x09, 0xd8, 0x6c, 0x83, 0x19, 0x93, 0xd4, 0xe2, 0x88, 0x88, 0x9b, 0xaf, 0x92,
      0x16, 0xc4, 0x15, 0xbd, 0x49, 0x13, 0x22, 0xb7, 0x84, 0xcf, 0x23, 0xf2, 0x6f,
      0x0c, 0x3e, 0x8f, 0xde, 0x04, 0x09, 0x31, 0x2d, 0x99, 0xdf, 0xe6, 0x74, 0x70,
      0x30, 0xde, 0x8c, 0xad, 0x32, 0x86, 0xe2, 0x7c, 0x12, 0x90, 0x21, 0xf3, 0x86,
      0xb7, 0xe2, 0x64, 0xca, 0x98, 0xcc, 0x64, 0x4b, 0xef, 0x57, 0x4f, 0x5a, 0x16,
      0x6e, 0xd7, 0x2f, 0x5b, 0xf6, 0x07, 0xad, 0x33, 0xb4, 0x8f, 0x3b, 0x3a, 0x8b,
      0xd9, 0x06, 0x2b, 0xed, 0x3c, 0x3c, 0x76, 0xf6, 0x21, 0x31, 0xe3, 0xfb, 0x2c,
      0x45, 0x61, 0x42, 0xba, 0xe0, 0xc3, 0x72, 0x63, 0xd0, 0x6b, 0x8f, 0x36, 0x26,
      0xfb, 0x9e, 0x89, 0x0e, 0x44, 0x9a, 0xc1, 0x84, 0x5e, 0x84, 0x8d, 0xb6, 0xea,
      0xf1, 0x0d, 0x66, 0xc7, 0xdb, 0x44, 0xbd, 0x19, 0x7c, 0x05, 0xbe, 0xc4, 0xab,
      0x88, 0x32, 0xbe, 0xc7, 0x63, 0x31, 0xe6, 0x38, 0xd4, 0xe5, 0xb8, 0x4b, 0xf5,
      0x0e, 0x55, 0x9a, 0x3a, 0xe6, 0x0a, 0xec, 0xee, 0xe2, 0xa8, 0x88, 0x04, 0xf2,
      0xb8, 0xaa, 0x5a, 0xd8, 0x97, 0x5d, 0xa0, 0xa8, 0x42, 0xfb, 0xd9, 0xde, 0x80,
      0xae, 0x4c, 0xb3, 0xa1, 0x90, 0x47, 0x57, 0x03, 0x10, 0x78, 0xa6, 0x8f, 0x11,
      0xba, 0x4b, 0xce, 0x2d, 0x56, 0xa4, 0xe1, 0xbd, 0xf8, 0xa0, 0xa4, 0xd5, 0x48,
      0x3c, 0x63, 0x20, 0x00, 0x38, 0xa0, 0xd1, 0xe6, 0x12, 0xe9, 0x1d, 0xd8, 0x49,
      0xe3, 0xd5, 0x24, 0xb5, 0xc5, 0x3a, 0x1f, 0xb0, 0xd4}};

const TPM2B_RSA_TEST_VALUE c_RsapssKvt =
```

```
    {RSA_TEST_KEY_SIZE,
     {0x74, 0x89, 0x29, 0x3e, 0x1b, 0xac, 0xc6, 0x85, 0xca, 0xf0, 0x63, 0x43, 0x30,
      0x7d, 0x1c, 0x9b, 0x2f, 0xbd, 0x4d, 0x69, 0x39, 0x5e, 0x85, 0xe2, 0xef, 0x86,
      0x0a, 0xc6, 0x6b, 0xa6, 0x08, 0x19, 0x6c, 0x56, 0x38, 0x24, 0x55, 0x92, 0x84,
      0x9b, 0x1b, 0x8b, 0x04, 0xcf, 0x24, 0x14, 0x24, 0x13, 0x0e, 0x8b, 0x82, 0x6f,
      0x96, 0xc8, 0x9a, 0x68, 0xfc, 0x4c, 0x02, 0xf0, 0xdc, 0xcd, 0x36, 0x25, 0x31,
      0xd5, 0x82, 0xcf, 0xc9, 0x69, 0x72, 0xf6, 0x1d, 0xab, 0x68, 0x20, 0x2e, 0x2d,
      0x19, 0x49, 0xf0, 0x2e, 0xad, 0xd2, 0xda, 0xaf, 0xff, 0xb6, 0x92, 0x83, 0x5b,
      0x8a, 0x06, 0x2d, 0x0c, 0x32, 0x11, 0x32, 0x3b, 0x77, 0x17, 0xf6, 0x50, 0xfb,
      0xf8, 0x57, 0xc9, 0xc7, 0x9b, 0x9e, 0xc6, 0xd1, 0xa9, 0x55, 0xf0, 0x22, 0x35,
      0xda, 0xca, 0x3c, 0x8e, 0xc6, 0x9a, 0xd8, 0x25, 0xc8, 0x5e, 0x93, 0x0d, 0xaa,
      0xa7, 0x06, 0xaf, 0x11, 0x29, 0x99, 0xe7, 0x7c, 0xee, 0x49, 0x82, 0x30, 0xba,
      0x2c, 0xe2, 0x40, 0x8f, 0x0a, 0xa6, 0x7b, 0x24, 0x75, 0xc5, 0xcd, 0x03, 0x12,
      0xf4, 0xb2, 0x4b, 0x3a, 0xd1, 0x91, 0x3c, 0x20, 0x0e, 0x58, 0x2b, 0x31, 0xf8,
      0x8b, 0xee, 0xbc, 0x1f, 0x95, 0x35, 0x58, 0x6a, 0x73, 0xee, 0x99, 0xb0, 0x01,
      0x42, 0x4f, 0x66, 0xc0, 0x66, 0xbb, 0x35, 0x86, 0xeb, 0xd9, 0x7b, 0x55, 0x77,
      0x2d, 0x54, 0x78, 0x19, 0x49, 0xe8, 0xcc, 0xfd, 0xb1, 0xcb, 0x49, 0xc9, 0xea,
      0x20, 0xab, 0xed, 0xb5, 0xed, 0xfe, 0xb2, 0xb5, 0xa8, 0xcf, 0x05, 0x06, 0xd5,
      0x7d, 0x2b, 0xbb, 0x0b, 0x65, 0x6b, 0x2b, 0x6d, 0x55, 0x95, 0x85, 0x44, 0x8b,
      0x12, 0x05, 0xf3, 0x4b, 0xd4, 0x8e, 0x3d, 0x68, 0x2d, 0x29, 0x9c, 0x05, 0x79,
      0xd6, 0xfc, 0x72, 0x90, 0x6a, 0xab, 0x46, 0x38, 0x81}};

const TPM2B_RSA_TEST_VALUE c_RsassaKvt =
    {RSA_TEST_KEY_SIZE,
     {0x8a, 0xb1, 0x0a, 0xb5, 0xe4, 0x02, 0xf7, 0xdd, 0x45, 0x2a, 0xcc, 0x2b, 0x6b,
      0x8c, 0x0e, 0x9a, 0x92, 0x4f, 0x9b, 0xc5, 0xe4, 0x8b, 0x82, 0xb9, 0xb0, 0xd9,
      0x87, 0x8c, 0xcb, 0xf0, 0xb0, 0x59, 0xa5, 0x92, 0x21, 0xa0, 0xa7, 0x61, 0x5c,
      0xed, 0xa8, 0x6e, 0x22, 0x29, 0x46, 0xc7, 0x86, 0x37, 0x4b, 0x1b, 0x1e, 0x94,
      0x93, 0xc8, 0x4c, 0x17, 0x7a, 0xae, 0x59, 0x91, 0xf8, 0x83, 0x84, 0xc4, 0x8c,
      0x38, 0xc2, 0x35, 0x0e, 0x7e, 0x50, 0x67, 0x76, 0xe7, 0xd3, 0xec, 0x6f, 0x0d,
      0xa0, 0x5c, 0x2f, 0x0a, 0x80, 0x28, 0xd3, 0xc5, 0x7d, 0x2d, 0x1a, 0x0b, 0x96,
      0xd6, 0xe5, 0x98, 0x05, 0x8c, 0x4d, 0xa0, 0x1f, 0x8c, 0xb6, 0xfb, 0xb1, 0xcf,
      0xe9, 0xcb, 0x38, 0x27, 0x60, 0x64, 0x17, 0xca, 0xf4, 0x8b, 0x61, 0xb7, 0x1d,
      0xb6, 0x20, 0x9d, 0x40, 0x2a, 0x1c, 0xfd, 0x55, 0x40, 0x4b, 0x95, 0x39, 0x52,
      0x18, 0x3b, 0xab, 0x44, 0xe8, 0x83, 0x4b, 0x7c, 0x47, 0xfb, 0xed, 0x06, 0x9c,
      0xcd, 0x4f, 0xba, 0x81, 0xd6, 0xb7, 0x31, 0xcf, 0x5c, 0x23, 0xf8, 0x25, 0xab,
      0x95, 0x77, 0x0a, 0x8f, 0x46, 0xef, 0xfb, 0x59, 0xb8, 0x04, 0xd7, 0x1e, 0xf5,
      0xaf, 0x6a, 0x1a, 0x26, 0x9b, 0xae, 0xf4, 0xf5, 0x7f, 0x84, 0x6f, 0x3c, 0xed,
      0xf8, 0x24, 0x0b, 0x43, 0xd1, 0xba, 0x74, 0x89, 0x4e, 0x39, 0xfe, 0xab, 0xa5,
      0x16, 0xa5, 0x28, 0xee, 0x96, 0x84, 0x3e, 0x16, 0x6d, 0x5f, 0x4e, 0x0b, 0x7d,
      0x94, 0x16, 0x1b, 0x8c, 0xf9, 0xaa, 0x9b, 0xc0, 0x49, 0x02, 0x4c, 0x3e, 0x62,
      0xff, 0xfe, 0xa2, 0x20, 0x33, 0x5e, 0xa6, 0xdd, 0xda, 0x15, 0x2d, 0xb7, 0xcd,
      0xda, 0xff, 0xb1, 0x0b, 0x45, 0x7b, 0xd3, 0xa0, 0x42, 0x29, 0xab, 0xa9, 0x73,
      0xe9, 0xa4, 0xd9, 0x8d, 0xac, 0xa1, 0x88, 0x2c, 0x2d}};

#endif  // SHA256

#if ALG_SHA384_VALUE == DEFAULT_TEST_HASH

const TPM2B_RSA_TEST_VALUE c_OaepKvt =
    {RSA_TEST_KEY_SIZE,
     {0x0f, 0x3c, 0x42, 0x4d, 0x8c, 0x91, 0x96, 0x05, 0x3c, 0xfd, 0x59, 0x3b, 0x7f,
      0x29, 0xbc, 0x03, 0x67, 0xc1, 0xff, 0x74, 0xe7, 0x09, 0xf4, 0x13, 0x45, 0xbe,
      0x13, 0x1d, 0xc9, 0x86, 0x94, 0xfe, 0xed, 0xa6, 0xe8, 0x3a, 0xcb, 0x89, 0x4d,
      0xec, 0x86, 0x63, 0x4c, 0xdb, 0xf1, 0x95, 0xee, 0xc1, 0x46, 0xc5, 0x3b, 0xd8,
      0xf8, 0xa2, 0x41, 0x6a, 0x60, 0x8b, 0x9e, 0x5e, 0x7f, 0x20, 0x16, 0xe3, 0x69,
      0xb6, 0x2d, 0x92, 0xfc, 0x60, 0xa2, 0x74, 0x88, 0xd5, 0xc7, 0xa6, 0xd1, 0xff,
      0xe3, 0x45, 0x02, 0x51, 0x39, 0xd9, 0xf3, 0x56, 0x0b, 0x91, 0x80, 0xe0, 0x6c,
      0xa8, 0xc3, 0x78, 0xef, 0x34, 0x22, 0x8c, 0xf5, 0xfb, 0x47, 0x98, 0x5d, 0x57,
      0x8e, 0x3a, 0xb9, 0xff, 0x92, 0x04, 0xc7, 0xc2, 0x6e, 0xfa, 0x14, 0xc1, 0xb9,
      0x68, 0x15, 0x5c, 0x12, 0xe8, 0xa8, 0xbe, 0xea, 0xe8, 0x8d, 0x9b, 0x48, 0x28,
      0x35, 0xdb, 0x4b, 0x52, 0xc1, 0x2d, 0x85, 0x47, 0x83, 0xd0, 0xe9, 0xae, 0x90,
      0x6e, 0x65, 0xd4, 0x34, 0x7f, 0x81, 0xce, 0x69, 0xf0, 0x96, 0x62, 0xf7, 0xec,
      0x41, 0xd5, 0xc2, 0xe3, 0x4b, 0xba, 0x9c, 0x8a, 0x02, 0xce, 0xf0, 0x5d, 0x14,
      0xf7, 0x09, 0x42, 0x8e, 0x4a, 0x27, 0xfe, 0x3e, 0x66, 0x42, 0x99, 0x03, 0xe1,
      0x69, 0xbd, 0xdb, 0x7f, 0x9b, 0x70, 0xeb, 0x4e, 0x9c, 0xac, 0x45, 0x67, 0x91,
```

```
      0x9f, 0x75, 0x10, 0xc6, 0xfc, 0x14, 0xe1, 0x28, 0xc1, 0x0e, 0xe0, 0x7e, 0xc0,
      0x5c, 0x1d, 0xee, 0xe8, 0xff, 0x45, 0x79, 0x51, 0x86, 0x08, 0xe6, 0x39, 0xac,
      0xb5, 0xfd, 0xb8, 0xf1, 0xdd, 0x2e, 0xf4, 0xb2, 0x1a, 0x69, 0x0d, 0xd9, 0x98,
      0x8e, 0xdb, 0x85, 0x61, 0x70, 0x20, 0x82, 0x91, 0x26, 0x87, 0x80, 0xc4, 0x6a,
      0xd8, 0x3b, 0x91, 0x4d, 0xd3, 0x33, 0x84, 0xad, 0xb7}};

const TPM2B_RSA_TEST_VALUE c_RsaesKvt =
    {RSA_TEST_KEY_SIZE,
     {0x44, 0xd5, 0x9f, 0xbc, 0x48, 0x03, 0x3d, 0x9f, 0x22, 0x91, 0x2a, 0xab, 0x3c,
      0x31, 0x71, 0xab, 0x86, 0x3f, 0x0f, 0x6f, 0x59, 0x5b, 0x93, 0x27, 0xbc, 0xbc,
      0xcd, 0x29, 0x38, 0x43, 0x2a, 0x3b, 0x3b, 0xd2, 0xb3, 0x45, 0x40, 0xba, 0x15,
      0xb4, 0x45, 0xe3, 0x56, 0xab, 0xff, 0xb3, 0x20, 0x26, 0x39, 0xcc, 0x48, 0xc5,
      0x5d, 0x41, 0x0d, 0x2f, 0x57, 0x7f, 0x9d, 0x16, 0x2e, 0x26, 0x57, 0xc7, 0x6b,
      0xf3, 0x36, 0x54, 0xbd, 0xb6, 0x1d, 0x46, 0x4e, 0x13, 0x50, 0xd7, 0x61, 0x9d,
      0x8d, 0x7b, 0xeb, 0x21, 0x9f, 0x79, 0xf3, 0xfd, 0xe0, 0x1b, 0xa8, 0xed, 0x6d,
      0x29, 0x33, 0x0d, 0x65, 0x94, 0x24, 0x1e, 0x62, 0x88, 0x6b, 0x2b, 0x4e, 0x39,
      0xf5, 0x80, 0x39, 0xca, 0x76, 0x95, 0xbc, 0x7c, 0x27, 0x1d, 0xdd, 0x3a, 0x11,
      0xf1, 0x3e, 0x54, 0x03, 0xb7, 0x43, 0x91, 0x99, 0x33, 0xfe, 0x9d, 0x14, 0x2c,
      0x87, 0x9a, 0x95, 0x18, 0x1f, 0x02, 0x04, 0x6a, 0xe2, 0xb7, 0x81, 0x14, 0x13,
      0x45, 0x16, 0xfb, 0xe4, 0xb7, 0x8f, 0xab, 0x2b, 0xd7, 0x60, 0x34, 0x8a, 0x55,
      0xbc, 0x01, 0x8c, 0x49, 0x02, 0x29, 0xf1, 0x9c, 0x94, 0x98, 0x44, 0xd0, 0x94,
      0xcb, 0xd4, 0x85, 0x4c, 0x3b, 0x77, 0x72, 0x99, 0xd5, 0x4b, 0xc6, 0x3b, 0xe4,
      0xd2, 0xc8, 0xe9, 0x6a, 0x23, 0x18, 0x3b, 0x3b, 0x5e, 0x32, 0xec, 0x70, 0x84,
      0x5d, 0xbb, 0x6a, 0x8f, 0x0c, 0x5f, 0x55, 0xa5, 0x30, 0x34, 0x48, 0xbb, 0xc2,
      0xdf, 0x12, 0xb9, 0x81, 0xad, 0x36, 0x3f, 0xf0, 0x24, 0x16, 0x48, 0x04, 0x4a,
      0x7f, 0xfd, 0x9f, 0x4c, 0xea, 0xfe, 0x1d, 0x83, 0xd0, 0x81, 0xad, 0x25, 0x6c,
      0x5f, 0x45, 0x36, 0x91, 0xf0, 0xd5, 0x8b, 0x53, 0x0a, 0xdf, 0xec, 0x9f, 0x04,
      0x58, 0xc4, 0x35, 0xa0, 0x78, 0x1f, 0x68, 0xe0, 0x22}};

const TPM2B_RSA_TEST_VALUE c_RsapssKvt =
    {RSA_TEST_KEY_SIZE,
     {0x3f, 0x3a, 0x82, 0x6d, 0x42, 0xe3, 0x8b, 0x4f, 0x45, 0x9c, 0xda, 0x6c, 0xbe,
      0xbe, 0xcd, 0x00, 0x98, 0xfb, 0xbe, 0x59, 0x30, 0xc6, 0x3c, 0xaa, 0xb3, 0x06,
      0x27, 0xb5, 0xda, 0xfa, 0xb2, 0xc3, 0x43, 0xb7, 0xbd, 0xe9, 0xd3, 0x23, 0xed,
      0x80, 0xce, 0x74, 0xb3, 0xb8, 0x77, 0x8d, 0xe6, 0x8d, 0x3c, 0xe5, 0xf5, 0xd7,
      0x80, 0xcf, 0x38, 0x55, 0x76, 0xd7, 0x87, 0xa8, 0xd6, 0x3a, 0xcf, 0xfd, 0xd8,
      0x91, 0x65, 0xab, 0x43, 0x66, 0x50, 0xb7, 0x9a, 0x13, 0x6b, 0x45, 0x80, 0x76,
      0x86, 0x22, 0x27, 0x72, 0xf7, 0xbb, 0x65, 0x22, 0x5c, 0x55, 0x60, 0xd8, 0x84,
      0x9f, 0xf2, 0x61, 0x52, 0xac, 0xf2, 0x4f, 0x5b, 0x7b, 0x21, 0xe1, 0xf5, 0x4b,
      0x8f, 0x01, 0xf2, 0x4b, 0xcf, 0xd3, 0xfb, 0x74, 0x5e, 0x6e, 0x96, 0xb4, 0xa8,
      0x0f, 0x01, 0x9b, 0x26, 0x54, 0x0a, 0x70, 0x55, 0x26, 0xb7, 0x0b, 0xe8, 0x01,
      0x68, 0x66, 0x0d, 0x6f, 0xb5, 0xfc, 0x66, 0xbd, 0x9e, 0x44, 0xed, 0x6a, 0x1e,
      0x3c, 0x3b, 0x61, 0x5d, 0xe8, 0xdb, 0x99, 0x5b, 0x67, 0xbf, 0x94, 0xfb, 0xe6,
      0x8c, 0x4b, 0x07, 0xcb, 0x43, 0x3a, 0x0d, 0xb1, 0x1b, 0x10, 0x66, 0x81, 0xe2,
      0x0d, 0xe7, 0xd1, 0xca, 0x85, 0xa7, 0x50, 0x82, 0x2d, 0xbf, 0xed, 0xcf, 0x43,
      0x6d, 0xdb, 0x2c, 0x7b, 0x73, 0x20, 0xfe, 0x73, 0x3f, 0x19, 0xc6, 0xdb, 0x69,
      0xb8, 0xc3, 0xd3, 0xf4, 0xe5, 0x64, 0xf8, 0x36, 0x8e, 0xd5, 0xd8, 0x09, 0x2a,
      0x5f, 0x26, 0x70, 0xa1, 0xd9, 0x5b, 0x14, 0xf8, 0x22, 0xe9, 0x9d, 0x22, 0x51,
      0xf4, 0x52, 0xc1, 0x6f, 0x53, 0xf5, 0xca, 0x0d, 0xda, 0x39, 0x8c, 0x29, 0x42,
      0xe8, 0x58, 0x89, 0xbb, 0xd1, 0x2e, 0xc5, 0xdb, 0x86, 0x8d, 0xaf, 0xec, 0x58,
      0x36, 0x8d, 0x8d, 0x57, 0x23, 0xd5, 0xdd, 0xb9, 0x24}};

const TPM2B_RSA_TEST_VALUE c_RsassaKvt =
    {RSA_TEST_KEY_SIZE,
     {0x39, 0x10, 0x58, 0x7d, 0x6d, 0xa8, 0xd5, 0x90, 0x07, 0xd6, 0x2b, 0x13, 0xe9,
      0xd8, 0x93, 0x7e, 0xf3, 0x5d, 0x71, 0xe0, 0xf0, 0x33, 0x3a, 0x4a, 0x22, 0xf3,
      0xe6, 0x95, 0xd3, 0x8e, 0x8c, 0x41, 0xe7, 0xb3, 0x13, 0xde, 0x4a, 0x45, 0xd3,
      0xd1, 0xfb, 0xb1, 0x3f, 0x9b, 0x39, 0xa5, 0x50, 0x58, 0xef, 0xb6, 0x3a, 0x43,
      0xdd, 0x54, 0xab, 0xda, 0x9d, 0x32, 0x49, 0xe4, 0x57, 0x96, 0xe5, 0x1b, 0x1d,
      0x8f, 0x33, 0x8e, 0x07, 0x67, 0x56, 0x14, 0xc1, 0x18, 0x78, 0xa2, 0x52, 0xe6,
      0x2e, 0x07, 0x81, 0xbe, 0xd8, 0xca, 0x76, 0x63, 0x68, 0xc5, 0x47, 0xa2, 0x92,
      0x5e, 0x4c, 0xfd, 0x14, 0xc7, 0x46, 0x14, 0xbe, 0xc7, 0x85, 0xef, 0xe6, 0xb8,
      0x46, 0xcb, 0x3a, 0x67, 0x66, 0x89, 0xc6, 0xee, 0x9d, 0x64, 0xf5, 0x0d, 0x09,
      0x80, 0x9a, 0x6f, 0x0e, 0xeb, 0xe4, 0xb9, 0xe9, 0xab, 0x90, 0x4f, 0xe7, 0x5a,
      0xc8, 0xca, 0xf6, 0x16, 0x0a, 0x82, 0xbd, 0xb7, 0x76, 0x59, 0x08, 0x2d, 0xd9,
      0x40, 0x5d, 0xaa, 0xa5, 0xef, 0xfb, 0xe3, 0x81, 0x2c, 0x2c, 0x5c, 0xa8, 0x16,
```

```
      0xbd, 0x63, 0x20, 0xc2, 0x4d, 0x3b, 0x51, 0xaa, 0x62, 0x1f, 0x06, 0xe5, 0xbb,
      0x78, 0x44, 0x04, 0x0c, 0x5c, 0xe1, 0x1b, 0x6b, 0x9d, 0x21, 0x10, 0xaf, 0x48,
      0x48, 0x98, 0x97, 0x77, 0xc2, 0x73, 0xb4, 0x98, 0x64, 0xcc, 0x94, 0x2c, 0x29,
      0x28, 0x45, 0x36, 0xd1, 0xc5, 0xd0, 0x2f, 0x97, 0x27, 0x92, 0x65, 0x22, 0xbb,
      0x63, 0x79, 0xea, 0xf5, 0xff, 0x77, 0x0f, 0x4b, 0x56, 0x8a, 0x9f, 0xad, 0x1a,
      0x97, 0x67, 0x39, 0x69, 0xb8, 0x4c, 0x6c, 0xc2, 0x56, 0xc5, 0x7a, 0xa8, 0x14,
      0x5a, 0x24, 0x7a, 0xa4, 0x6e, 0x55, 0xb2, 0x86, 0x1d, 0xf4, 0x62, 0x5a, 0x2d,
      0x87, 0x6d, 0xde, 0x99, 0x78, 0x2d, 0xef, 0xd7, 0xdc}};

#endif  // SHA384

#if ALG_SHA512_VALUE == DEFAULT_TEST_HASH

const TPM2B_RSA_TEST_VALUE c_OaepKvt =
    {RSA_TEST_KEY_SIZE,
     {0x48, 0x45, 0xa7, 0x70, 0xb2, 0x41, 0xb7, 0x48, 0x5e, 0x79, 0x8c, 0xdf, 0x1c,
      0xc6, 0x7e, 0xbb, 0x11, 0x80, 0x82, 0x52, 0xbf, 0x40, 0x3d, 0x90, 0x03, 0x6e,
      0x20, 0x3a, 0xb9, 0x65, 0xc8, 0x51, 0x4c, 0xbd, 0x9c, 0xa9, 0x43, 0x89, 0xd0,
      0x57, 0x0c, 0xa3, 0x69, 0x22, 0x7e, 0x82, 0x2a, 0x1c, 0x1d, 0x5a, 0x80, 0x84,
      0x81, 0xbb, 0x5e, 0x5e, 0xd0, 0xc1, 0x66, 0x9a, 0xac, 0x00, 0xba, 0x14, 0xa2,
      0xe9, 0xd0, 0x3a, 0x89, 0x5a, 0x63, 0xe2, 0xec, 0x92, 0x05, 0xf4, 0x47, 0x66,
      0x12, 0x7f, 0xdb, 0xa7, 0x3c, 0x5b, 0x67, 0xe1, 0x55, 0xca, 0x0a, 0x27, 0xbf,
      0x39, 0x89, 0x11, 0x05, 0xba, 0x9b, 0x5a, 0x9b, 0x65, 0x44, 0xad, 0x78, 0xcf,
      0x8f, 0x94, 0xf6, 0x9a, 0xb4, 0x52, 0x39, 0x0e, 0x00, 0xba, 0xbc, 0xe0, 0xbd,
      0x6f, 0x81, 0x2d, 0x76, 0x42, 0x66, 0x70, 0x07, 0x77, 0xbf, 0x09, 0x88, 0x2a,
      0x0c, 0xb1, 0x56, 0x3e, 0xee, 0xfd, 0xdc, 0xb6, 0x3c, 0x0d, 0xc5, 0xa4, 0x0d,
      0x10, 0x32, 0x80, 0x3e, 0x1e, 0xfe, 0x36, 0x8f, 0xb5, 0x42, 0xc1, 0x21, 0x7b,
      0xdf, 0xdf, 0x4a, 0xd2, 0x68, 0x0c, 0x01, 0x9f, 0x4a, 0xfd, 0xd4, 0xec, 0xf7,
      0x49, 0x06, 0xab, 0xed, 0xc6, 0xd5, 0x1b, 0x63, 0x76, 0x38, 0xc8, 0x6c, 0xc7,
      0x4f, 0xcb, 0x29, 0x8a, 0x0e, 0x6f, 0x33, 0xaf, 0x69, 0x31, 0x8e, 0xa7, 0xdd,
      0x9a, 0x36, 0xde, 0x9b, 0xf1, 0x0b, 0xfb, 0x20, 0xa0, 0x6d, 0x33, 0x31, 0xc9,
      0x9e, 0xb4, 0x2e, 0xc5, 0x40, 0x0e, 0x60, 0x71, 0x36, 0x75, 0x05, 0xf9, 0x37,
      0xe0, 0xca, 0x8e, 0x8f, 0x56, 0xe0, 0xea, 0x9b, 0xeb, 0x17, 0xf3, 0xca, 0x40,
      0xc3, 0x48, 0x01, 0xba, 0xdc, 0xc6, 0x4b, 0x2b, 0x5b, 0x7b, 0x5c, 0x81, 0xa6,
      0xbb, 0xc7, 0x43, 0xc0, 0xbe, 0xc0, 0x30, 0x7b, 0x55}};

const TPM2B_RSA_TEST_VALUE c_RsaesKvt =
    {RSA_TEST_KEY_SIZE,
     {0x74, 0x83, 0xfa, 0x52, 0x65, 0x50, 0x68, 0xd0, 0x82, 0x05, 0x72, 0x70, 0x78,
      0x1c, 0xac, 0x10, 0x23, 0xc5, 0x07, 0xf8, 0x93, 0xd2, 0xeb, 0x65, 0x87, 0xbb,
      0x47, 0xc2, 0xfb, 0x30, 0x9e, 0x61, 0x4c, 0xac, 0x04, 0x57, 0x5a, 0x7c, 0xeb,
      0x29, 0x08, 0x84, 0x86, 0x89, 0x1e, 0x8f, 0x07, 0x32, 0xa3, 0x8b, 0x70, 0xe7,
      0xa2, 0x9f, 0x9c, 0x42, 0x71, 0x3d, 0x23, 0x59, 0x82, 0x5e, 0x8a, 0xde, 0xd6,
      0xfb, 0xd8, 0xc5, 0x8b, 0xc0, 0xdb, 0x10, 0x38, 0x87, 0xd3, 0xbf, 0x04, 0xb0,
      0x66, 0xb9, 0x85, 0x81, 0x54, 0x4c, 0x69, 0xdc, 0xba, 0x78, 0xf3, 0x4a, 0xdb,
      0x25, 0xa2, 0xf2, 0x34, 0x55, 0xdd, 0xaa, 0xa5, 0xc4, 0xed, 0x55, 0x06, 0x0e,
      0x2a, 0x30, 0x77, 0xab, 0x82, 0x79, 0xf0, 0xcd, 0x9d, 0x6f, 0x09, 0xa0, 0xc8,
      0x82, 0xc9, 0xe0, 0x61, 0xda, 0x40, 0xcd, 0x17, 0x59, 0xc0, 0xef, 0x95, 0x6d,
      0xa3, 0x6d, 0x1c, 0x2b, 0xee, 0x24, 0xef, 0xd8, 0x4a, 0x55, 0x6c, 0xd6, 0x26,
      0x42, 0x32, 0x17, 0xfd, 0x6a, 0xb3, 0x4f, 0xde, 0x07, 0x2f, 0x10, 0xd4, 0xac,
      0x14, 0xea, 0x89, 0x68, 0xcc, 0xd3, 0x07, 0xb7, 0xcf, 0xba, 0x39, 0x20, 0x63,
      0x20, 0x7b, 0x44, 0x8b, 0x48, 0x60, 0x5d, 0x3a, 0x2a, 0x0a, 0xe9, 0x68, 0xab,
      0x15, 0x46, 0x27, 0x64, 0xb5, 0x82, 0x06, 0x29, 0xe7, 0x25, 0xca, 0x46, 0x48,
      0x6e, 0x2a, 0x34, 0x57, 0x4b, 0x81, 0x75, 0xae, 0xb6, 0xfd, 0x6f, 0x51, 0x5f,
      0x04, 0x59, 0xc7, 0x15, 0x1f, 0xe0, 0x68, 0xf7, 0x36, 0x2d, 0xdf, 0xc8, 0x9d,
      0x05, 0x27, 0x2d, 0x3f, 0x2b, 0x59, 0x5d, 0xcb, 0xf3, 0xc4, 0x92, 0x6e, 0x00,
      0xa8, 0x8d, 0xd0, 0x69, 0xe5, 0x59, 0xda, 0xba, 0x4f, 0x38, 0xf5, 0xa0, 0x8b,
      0xf1, 0x73, 0xe9, 0x0d, 0xee, 0x64, 0xe5, 0xa2, 0xd8}};

const TPM2B_RSA_TEST_VALUE c_RsapssKvt =
    {RSA_TEST_KEY_SIZE,
     {0x1b, 0xca, 0x8b, 0x18, 0x15, 0x3b, 0x95, 0x5b, 0x0a, 0x89, 0x10, 0x03, 0x7f,
      0x7c, 0xa0, 0xc9, 0x66, 0x57, 0x86, 0x6a, 0xc9, 0xeb, 0x82, 0x71, 0xf3, 0x8d,
      0x6f, 0xa9, 0xa4, 0x2d, 0xd0, 0x22, 0xdf, 0xe9, 0xc6, 0x71, 0x5b, 0xf4, 0x27,
      0x38, 0x5b, 0x2c, 0x8a, 0x54, 0xcc, 0x85, 0x11, 0x69, 0x6d, 0x6f, 0x42, 0xe7,
      0x22, 0xcb, 0xd6, 0xad, 0x1a, 0xc5, 0xab, 0x6a, 0xa5, 0xfc, 0xa5, 0x70, 0x72,
```

```
      0x4a, 0x62, 0x25, 0xd0, 0xa2, 0x16, 0x61, 0xab, 0xac, 0x31, 0xa0, 0x46, 0x24,
      0x4f, 0xdd, 0x9a, 0x36, 0x55, 0xb6, 0x00, 0x9e, 0x23, 0x50, 0x0d, 0x53, 0x01,
      0xb3, 0x46, 0x56, 0xb2, 0x1d, 0x33, 0x5b, 0xca, 0x41, 0x7f, 0x65, 0x7e, 0x00,
      0x5c, 0x12, 0xff, 0x0a, 0x70, 0x5d, 0x8c, 0x69, 0x4a, 0x02, 0xee, 0x72, 0x30,
      0xa7, 0x5c, 0xa4, 0xbb, 0xbe, 0x03, 0x0c, 0xe4, 0x5f, 0x33, 0xb6, 0x78, 0x91,
      0x9d, 0xd8, 0xec, 0x34, 0x03, 0x2e, 0x63, 0x32, 0xc7, 0x2a, 0x36, 0x50, 0xd5,
      0x8b, 0x0e, 0x7f, 0x54, 0x4e, 0xf4, 0x29, 0x11, 0x1b, 0xcd, 0x0f, 0x37, 0xa5,
      0xbc, 0x61, 0x83, 0x50, 0xfa, 0x18, 0x75, 0xd9, 0xfe, 0xa7, 0xe8, 0x9b, 0xc1,
      0x4f, 0x96, 0x37, 0x81, 0x71, 0xdf, 0x71, 0x8b, 0x89, 0x81, 0xf4, 0x95, 0xb5,
      0x29, 0x66, 0x41, 0x0c, 0x73, 0xd7, 0x0b, 0x21, 0xb4, 0xfb, 0xf9, 0x63, 0x2f,
      0xe9, 0x7b, 0x38, 0xaa, 0x20, 0xc3, 0x96, 0xcc, 0xb7, 0xb2, 0x24, 0xa1, 0xe0,
      0x59, 0x9c, 0x10, 0x9e, 0x5a, 0xf7, 0xe3, 0x02, 0xe6, 0x23, 0xe2, 0x44, 0x21,
      0x3f, 0x6e, 0x5e, 0x79, 0xb2, 0x93, 0x7d, 0xce, 0xed, 0xe2, 0xe1, 0xab, 0x98,
      0x07, 0xa7, 0xbd, 0xbc, 0xd8, 0xf7, 0x06, 0xeb, 0xc5, 0xa6, 0x37, 0x18, 0x11,
      0x88, 0xf7, 0x63, 0x39, 0xb9, 0x57, 0x29, 0xdc, 0x03}};

const TPM2B_RSA_TEST_VALUE c_RsassaKvt =
    {RSA_TEST_KEY_SIZE,
     {0x05, 0x55, 0x00, 0x62, 0x01, 0xc6, 0x04, 0x31, 0x55, 0x73, 0x3f, 0x2a, 0xf9,
      0xd4, 0x0f, 0xc1, 0x2b, 0xeb, 0xd8, 0xc8, 0xdb, 0xb2, 0xab, 0x6c, 0x26, 0xde,
      0x2d, 0x89, 0xc2, 0x2d, 0x36, 0x62, 0xc8, 0x22, 0x5d, 0x58, 0x03, 0xb1, 0x46,
      0x14, 0xa5, 0xd4, 0xbc, 0x25, 0x6b, 0x7f, 0x8f, 0x14, 0x7e, 0x03, 0x2f, 0x3d,
      0xb8, 0x39, 0xa5, 0x79, 0x13, 0x7e, 0x22, 0x2a, 0xb9, 0x3e, 0x8f, 0xaa, 0x01,
      0x7c, 0x03, 0x12, 0x21, 0x6c, 0x2a, 0xb4, 0x39, 0x98, 0x6d, 0xff, 0x08, 0x6c,
      0x59, 0x2d, 0xdc, 0xc6, 0xf1, 0x77, 0x62, 0x10, 0xa6, 0xcc, 0xe2, 0x71, 0x8e,
      0x97, 0x00, 0x87, 0x5b, 0x0e, 0x20, 0x00, 0x3f, 0x18, 0x63, 0x83, 0xf0, 0xe4,
      0x0a, 0x64, 0x8c, 0xe9, 0x8c, 0x91, 0xe7, 0x89, 0x04, 0x64, 0x2c, 0x8b, 0x41,
      0xc8, 0xac, 0xf6, 0x5a, 0x75, 0xe6, 0xa5, 0x76, 0x43, 0xcb, 0xa5, 0x33, 0x8b,
      0x07, 0xc9, 0x73, 0x0f, 0x45, 0xa4, 0xc3, 0xac, 0xc1, 0xc3, 0xe6, 0xe7, 0x21,
      0x66, 0x1c, 0xba, 0xbf, 0xea, 0x3e, 0x39, 0xfa, 0xb2, 0xe2, 0x8f, 0xfe, 0x9c,
      0xb4, 0x85, 0x89, 0x33, 0x2a, 0x0c, 0xc8, 0x5d, 0x58, 0xe1, 0x89, 0x12, 0xe9,
      0x4d, 0x42, 0xb3, 0x1f, 0x99, 0x0c, 0x3e, 0xd8, 0xb2, 0xeb, 0xf5, 0x88, 0xfb,
      0xe1, 0x4b, 0x8e, 0xdc, 0xd3, 0xa8, 0xda, 0xbe, 0x04, 0x45, 0xbf, 0x56, 0xc6,
      0x54, 0x70, 0x00, 0xb8, 0x66, 0x46, 0x3a, 0xa3, 0x1e, 0xb6, 0xeb, 0x1a, 0xa0,
      0x0b, 0xd3, 0x9a, 0x9a, 0x52, 0xda, 0x60, 0x69, 0xb7, 0xef, 0x93, 0x47, 0x38,
      0xab, 0x1a, 0xa0, 0x22, 0x6e, 0x76, 0x06, 0xb6, 0x74, 0xaf, 0x74, 0x8f, 0x51,
      0xc0, 0x89, 0x5a, 0x4b, 0xbe, 0x6a, 0x91, 0x18, 0x25, 0x7d, 0xa6, 0x77, 0xe6,
      0xfd, 0xc2, 0x62, 0x36, 0x07, 0xc6, 0xef, 0x79, 0xc9}};

#endif  // SHA512
```

6.34 /tpm/include/private/SelfTest.h

```
//** Introduction
// This file contains the structure definitions for the self-test. It also contains
// macros for use when the self-test is implemented.
#ifndef _SELF_TEST_H_
#define _SELF_TEST_H_

//** Defines

// Was typing this a lot
#define SELF_TEST_FAILURE FAIL(FATAL_ERROR_SELF_TEST)

// Use the definition of key sizes to set algorithm values for key size.
#define AES_ENTRIES      (AES_128 + AES_192 + AES_256)
#define SM4_ENTRIES      (SM4_128)
#define CAMELLIA_ENTRIES (CAMELLIA_128 + CAMELLIA_192 + CAMELLIA_256)

#define NUM_SYMS (AES_ENTRIES + SM4_ENTRIES + CAMELLIA_ENTRIES)

typedef UINT32 SYM_INDEX;

// These two defines deal with the fact that the TPM_ALG_ID table does not delimit
// the symmetric mode values with a SYM_MODE_FIRST and SYM_MODE_LAST
```

```
#define SYM_MODE_FIRST ALG_CTR_VALUE
#define SYM_MODE_LAST  ALG_ECB_VALUE

#define NUM_SYM_MODES (SYM_MODE_LAST - SYM_MODE_FIRST + 1)

// Define a type to hold a bit vector for the modes.
#if NUM_SYM_MODES <= 0
#  error "No symmetric modes implemented"
#elif NUM_SYM_MODES <= 8
typedef BYTE SYM_MODES;
#elif NUM_SYM_MODES <= 16
typedef UINT16 SYM_MODES;
#elif NUM_SYM_MODES <= 32
typedef UINT32 SYM_MODES;
#else
#  error "Too many symmetric modes"
#endif

typedef struct SYMMETRIC_TEST_VECTOR
{
    const TPM_ALG_ID alg;                    // the algorithm
    const UINT16     keyBits;                // bits in the key
    const BYTE*      key;                    // The test key
    const UINT32     ivSize;                 // block size of the algorithm
    const UINT32     dataInOutSize;          // size  to encrypt/decrypt
    const BYTE*      dataIn;                 // data to encrypt
    const BYTE*      dataOut[NUM_SYM_MODES]; // data to decrypt
} SYMMETRIC_TEST_VECTOR;

#if ALG_SHA512
#  define DEFAULT_TEST_HASH             ALG_SHA512_VALUE
#  define DEFAULT_TEST_DIGEST_SIZE      SHA512_DIGEST_SIZE
#  define DEFAULT_TEST_HASH_BLOCK_SIZE SHA512_BLOCK_SIZE
#elif ALG_SHA384
#  define DEFAULT_TEST_HASH             ALG_SHA384_VALUE
#  define DEFAULT_TEST_DIGEST_SIZE      SHA384_DIGEST_SIZE
#  define DEFAULT_TEST_HASH_BLOCK_SIZE SHA384_BLOCK_SIZE
#elif ALG_SHA256
#  define DEFAULT_TEST_HASH             ALG_SHA256_VALUE
#  define DEFAULT_TEST_DIGEST_SIZE      SHA256_DIGEST_SIZE
#  define DEFAULT_TEST_HASH_BLOCK_SIZE SHA256_BLOCK_SIZE
#elif ALG_SHA1
#  define DEFAULT_TEST_HASH             ALG_SHA1_VALUE
#  define DEFAULT_TEST_DIGEST_SIZE      SHA1_DIGEST_SIZE
#  define DEFAULT_TEST_HASH_BLOCK_SIZE SHA1_BLOCK_SIZE
#endif

#endif  // _SELF_TEST_H_
```

6.35 /tpm/include/private/SymmetricTest.h

```
//** Introduction

// This file contains the structures and data definitions for the symmetric tests.
// This file references the header file that contains the actual test vectors. This
// organization was chosen so that the program that is used to generate the test
// vector values does not have to also re-generate this data.
#ifndef SELF_TEST_DATA
#  error "This file may only be included in AlgorithmTests.c"
#endif

#ifndef _SYMMETRIC_TEST_H
#  define _SYMMETRIC_TEST_H
#  include "SymmetricTestData.h"
```

```
//** Symmetric Test Structures

const SYMMETRIC_TEST_VECTOR c_symTestValues[NUM_SYMS + 1] = {
#  if ALG_AES && AES_128
    {TPM_ALG_AES,
     128,
     key_AES128,
     16,
     sizeof(dataIn_AES128),
     dataIn_AES128,
     {dataOut_AES128_CTR,
      dataOut_AES128_OFB,
      dataOut_AES128_CBC,
      dataOut_AES128_CFB,
      dataOut_AES128_ECB}},
#  endif
#  if ALG_AES && AES_192
    {TPM_ALG_AES,
     192,
     key_AES192,
     16,
     sizeof(dataIn_AES192),
     dataIn_AES192,
     {dataOut_AES192_CTR,
      dataOut_AES192_OFB,
      dataOut_AES192_CBC,
      dataOut_AES192_CFB,
      dataOut_AES192_ECB}},
#  endif
#  if ALG_AES && AES_256
    {TPM_ALG_AES,
     256,
     key_AES256,
     16,
     sizeof(dataIn_AES256),
     dataIn_AES256,
     {dataOut_AES256_CTR,
      dataOut_AES256_OFB,
      dataOut_AES256_CBC,
      dataOut_AES256_CFB,
      dataOut_AES256_ECB}},
#  endif
// There are no SM4 test values yet so...
#  if ALG_SM4 && SM4_128 && 0
    {TPM_ALG_SM4,
     128,
     key_SM4128,
     16,
     sizeof(dataIn_SM4128),
     dataIn_SM4128,
     {dataOut_SM4128_CTR,
      dataOut_SM4128_OFB,
      dataOut_SM4128_CBC,
      dataOut_SM4128_CFB,
      dataOut_AES128_ECB}},
#  endif
    {0}};

#endif  // _SYMMETRIC_TEST_H
```

6.36 /tpm/include/private/SymmetricTestData.h

```
// This is a vector for testing either encrypt or decrypt. The premise for decrypt
// is that the IV for decryption is the same as the IV for encryption. However,
// the ivOut value may be different for encryption and decryption. We will encrypt
```

```
// at least two blocks. This means that the chaining value will be used for each
// of the schemes (if any) and that implicitly checks that the chaining value
// is handled properly.

#if AES_128

const BYTE key_AES128[]         = {0x2b,
                                   0x7e,
                                   0x15,
                                   0x16,
                                   0x28,
                                   0xae,
                                   0xd2,
                                   0xa6,
                                   0xab,
                                   0xf7,
                                   0x15,
                                   0x88,
                                   0x09,
                                   0xcf,
                                   0x4f,
                                   0x3c};

const BYTE dataIn_AES128[]      = {0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
                                   0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                   0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
                                   0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};

const BYTE dataOut_AES128_ECB[] = {0x3a, 0xd7, 0x7b, 0xb4, 0x0d, 0x7a, 0x36, 0x60,
                                   0xa8, 0x9e, 0xca, 0xf3, 0x24, 0x66, 0xef, 0x97,
                                   0xf5, 0xd3, 0xd5, 0x85, 0x03, 0xb9, 0x69, 0x9d,
                                   0xe7, 0x85, 0x89, 0x5a, 0x96, 0xfd, 0xba, 0xaf};

const BYTE dataOut_AES128_CBC[] = {0x76, 0x49, 0xab, 0xac, 0x81, 0x19, 0xb2, 0x46,
                                   0xce, 0xe9, 0x8e, 0x9b, 0x12, 0xe9, 0x19, 0x7d,
                                   0x50, 0x86, 0xcb, 0x9b, 0x50, 0x72, 0x19, 0xee,
                                   0x95, 0xdb, 0x11, 0x3a, 0x91, 0x76, 0x78, 0xb2};

const BYTE dataOut_AES128_CFB[] = {0x3b, 0x3f, 0xd9, 0x2e, 0xb7, 0x2d, 0xad, 0x20,
                                   0x33, 0x34, 0x49, 0xf8, 0xe8, 0x3c, 0xfb, 0x4a,
                                   0xc8, 0xa6, 0x45, 0x37, 0xa0, 0xb3, 0xa9, 0x3f,
                                   0xcd, 0xe3, 0xcd, 0xad, 0x9f, 0x1c, 0xe5, 0x8b};

const BYTE dataOut_AES128_OFB[] = {0x3b, 0x3f, 0xd9, 0x2e, 0xb7, 0x2d, 0xad, 0x20,
                                   0x33, 0x34, 0x49, 0xf8, 0xe8, 0x3c, 0xfb, 0x4a,
                                   0x77, 0x89, 0x50, 0x8d, 0x16, 0x91, 0x8f, 0x03,
                                   0xf5, 0x3c, 0x52, 0xda, 0xc5, 0x4e, 0xd8, 0x25};

const BYTE dataOut_AES128_CTR[] = {0x87, 0x4d, 0x61, 0x91, 0xb6, 0x20, 0xe3, 0x26,
                                   0x1b, 0xef, 0x68, 0x64, 0x99, 0x0d, 0xb6, 0xce,
                                   0x98, 0x06, 0xf6, 0x6b, 0x79, 0x70, 0xfd, 0xff,
                                   0x86, 0x17, 0x18, 0x7b, 0xb9, 0xff, 0xfd, 0xff};
#endif

#if AES_192

const BYTE key_AES192[]         = {0x8e, 0x73, 0xb0, 0xf7, 0xda, 0x0e, 0x64, 0x52,
                                   0xc8, 0x10, 0xf3, 0x2b, 0x80, 0x90, 0x79, 0xe5,
                                   0x62, 0xf8, 0xea, 0xd2, 0x52, 0x2c, 0x6b, 0x7b};

const BYTE dataIn_AES192[]      = {0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
                                   0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                   0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
                                   0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};

const BYTE dataOut_AES192_ECB[] = {0xbd, 0x33, 0x4f, 0x1d, 0x6e, 0x45, 0xf2, 0x5f,
```

```
                                    0xf7, 0x12, 0xa2, 0x14, 0x57, 0x1f, 0xa5, 0xcc,
                                    0x97, 0x41, 0x04, 0x84, 0x6d, 0x0a, 0xd3, 0xad,
                                    0x77, 0x34, 0xec, 0xb3, 0xec, 0xee, 0x4e, 0xef};

const BYTE dataOut_AES192_CBC[] = {0x4f, 0x02, 0x1d, 0xb2, 0x43, 0xbc, 0x63, 0x3d,
                                    0x71, 0x78, 0x18, 0x3a, 0x9f, 0xa0, 0x71, 0xe8,
                                    0xb4, 0xd9, 0xad, 0xa9, 0xad, 0x7d, 0xed, 0xf4,
                                    0xe5, 0xe7, 0x38, 0x76, 0x3f, 0x69, 0x14, 0x5a};

const BYTE dataOut_AES192_CFB[] = {0xcd, 0xc8, 0x0d, 0x6f, 0xdd, 0xf1, 0x8c, 0xab,
                                    0x34, 0xc2, 0x59, 0x09, 0xc9, 0x9a, 0x41, 0x74,
                                    0x67, 0xce, 0x7f, 0x7f, 0x81, 0x17, 0x36, 0x21,
                                    0x96, 0x1a, 0x2b, 0x70, 0x17, 0x1d, 0x3d, 0x7a};

const BYTE dataOut_AES192_OFB[] = {0xcd, 0xc8, 0x0d, 0x6f, 0xdd, 0xf1, 0x8c, 0xab,
                                    0x34, 0xc2, 0x59, 0x09, 0xc9, 0x9a, 0x41, 0x74,
                                    0xfc, 0xc2, 0x8b, 0x8d, 0x4c, 0x63, 0x83, 0x7c,
                                    0x09, 0xe8, 0x17, 0x00, 0xc1, 0x10, 0x04, 0x01};

const BYTE dataOut_AES192_CTR[] = {0x1a, 0xbc, 0x93, 0x24, 0x17, 0x52, 0x1c, 0xa2,
                                    0x4f, 0x2b, 0x04, 0x59, 0xfe, 0x7e, 0x6e, 0x0b,
                                    0x09, 0x03, 0x39, 0xec, 0x0a, 0xa6, 0xfa, 0xef,
                                    0xd5, 0xcc, 0xc2, 0xc6, 0xf4, 0xce, 0x8e, 0x94};
#endif

#if AES_256

const BYTE key_AES256[]         = {0x60, 0x3d, 0xeb, 0x10, 0x15, 0xca, 0x71, 0xbe,
                                    0x2b, 0x73, 0xae, 0xf0, 0x85, 0x7d, 0x77, 0x81,
                                    0x1f, 0x35, 0x2c, 0x07, 0x3b, 0x61, 0x08, 0xd7,
                                    0x2d, 0x98, 0x10, 0xa3, 0x09, 0x14, 0xdf, 0xf4};

const BYTE dataIn_AES256[]      = {0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
                                    0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
                                    0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
                                    0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};

const BYTE dataOut_AES256_ECB[] = {0xf3, 0xee, 0xd1, 0xbd, 0xb5, 0xd2, 0xa0, 0x3c,
                                    0x06, 0x4b, 0x5a, 0x7e, 0x3d, 0xb1, 0x81, 0xf8,
                                    0x59, 0x1c, 0xcb, 0x10, 0xd4, 0x10, 0xed, 0x26,
                                    0xdc, 0x5b, 0xa7, 0x4a, 0x31, 0x36, 0x28, 0x70};

const BYTE dataOut_AES256_CBC[] = {0xf5, 0x8c, 0x4c, 0x04, 0xd6, 0xe5, 0xf1, 0xba,
                                    0x77, 0x9e, 0xab, 0xfb, 0x5f, 0x7b, 0xfb, 0xd6,
                                    0x9c, 0xfc, 0x4e, 0x96, 0x7e, 0xdb, 0x80, 0x8d,
                                    0x67, 0x9f, 0x77, 0x7b, 0xc6, 0x70, 0x2c, 0x7d};

const BYTE dataOut_AES256_CFB[] = {0xdc, 0x7e, 0x84, 0xbf, 0xda, 0x79, 0x16, 0x4b,
                                    0x7e, 0xcd, 0x84, 0x86, 0x98, 0x5d, 0x38, 0x60,
                                    0x39, 0xff, 0xed, 0x14, 0x3b, 0x28, 0xb1, 0xc8,
                                    0x32, 0x11, 0x3c, 0x63, 0x31, 0xe5, 0x40, 0x7b};

const BYTE dataOut_AES256_OFB[] = {0xdc, 0x7e, 0x84, 0xbf, 0xda, 0x79, 0x16, 0x4b,
                                    0x7e, 0xcd, 0x84, 0x86, 0x98, 0x5d, 0x38, 0x60,
                                    0x4f, 0xeb, 0xdc, 0x67, 0x40, 0xd2, 0x0b, 0x3a,
                                    0xc8, 0x8f, 0x6a, 0xd8, 0x2a, 0x4f, 0xb0, 0x8d};

const BYTE dataOut_AES256_CTR[] = {0x60, 0x1e, 0xc3, 0x13, 0x77, 0x57, 0x89, 0xa5,
                                    0xb7, 0xa7, 0xf5, 0x04, 0xbb, 0xf3, 0xd2, 0x28,
                                    0xf4, 0x43, 0xe3, 0xca, 0x4d, 0x62, 0xb5, 0x9a,
                                    0xca, 0x84, 0xe9, 0x90, 0xca, 0xca, 0xf5, 0xc5};
#endif
```

6.37 /tpm/include/private/TableMarshal.h

```
#ifndef _TABLE_MARSHAL_H_
#define _TABLE_MARSHAL_H_

// These are the basic unmarshaling types. This is in the first byte of
// each structure descriptor that is passed to Marshal()/Unmarshal() for processing.
#define UINT_MTYPE       0
#define VALUES_MTYPE     (UINT_MTYPE + 1)
#define TABLE_MTYPE      (VALUES_MTYPE + 1)
#define MIN_MAX_MTYPE    (TABLE_MTYPE + 1)
#define ATTRIBUTES_MTYPE (MIN_MAX_MTYPE + 1)
#define STRUCTURE_MTYPE  (ATTRIBUTES_MTYPE + 1)
#define TPM2B_MTYPE      (STRUCTURE_MTYPE + 1)
#define TPM2BS_MTYPE     (TPM2B_MTYPE + 1)
#define LIST_MTYPE       (TPM2BS_MTYPE + 1)  // TPML
#define ERROR_MTYPE      (LIST_MTYPE + 1)
#define NULL_MTYPE       (ERROR_MTYPE + 1)
#define COMPOSITE_MTYPE  (NULL_MTYPE + 1)

//*** The Marshal Index
// A structure is used to hold the values that guide the marshaling/unmarshaling of
// each of the types. Each structure has a name and an address. For a structure to
// define a TPMS_name, the structure is a TPMS_name_MARSHAL_STRUCT and its
// index is TPMS_name_MARSHAL_INDEX. So, to get the proper structure, use the
// associated marshal index. The marshal index is passed to Marshal() or Unmarshal()
// and those functions look up the proper structure.
//
// To handle structures that allow a null value, the upper bit of each marshal
// index indicates if the null value is allowed. This is the NULL_FLAG. It is defined
// in TableMarshalIndex.h because it is needed by code outside of the marshaling
// code.

// A structure will have a list of marshal indexes to indicate what to unmarshal. When
// that index appears in a structure/union, the value will contain a flag to indicate
// that the NULL_FLAG should be SET on the call to Unmarshal() to unmarshal the type.
// The caller simply takes the entry and passes it to Unmarshal() to indicate that the
// NULL_FLAG is SET. There is also the opportunity to SET the NULL_FLAG in the called
// structure if the NULL_FLAG was set in the call to the calling structure. This is
// indicated by:
#define NULL_MASK ~(NULL_FLAG)

// When looking up the value to marshal, the upper bit of the marshal index is
// masked to yield the actual index. The MSb is the flag bit that indicates if a
// null flag is set. Code does not verify that the bit is clear when the called object
// does not take a flag as this is a benign error.

// the modifier byte as used by each MTYPE shown as a structure. They are expressed
// as a bit maps below. However, the code uses masking and not bit fields. The types
// show below are just to help in understanding.
// NOTE: LSb0 bit numbering is assumed in these typedefs.
//
// When used in an UINT_MTYPE
typedef struct integerModifier
{
    unsigned size   : 2;
    unsigned sign   : 1;
    unsigned unused : 7;
} integerModifier;

// When used in a VALUES_MTYPE
typedef struct valuesModifier
{
    unsigned size      : 2;
    unsigned sign      : 1;
    unsigned unused    : 5;
```

```
    unsigned takesNull : 1;
} valuesModifier;

// When used in a TABLE_MTYPE
typedef struct tableModifier
{
    unsigned size      : 2;
    unsigned sign      : 1;
    unsigned unused    : 3;
    unsigned hasBits   : 1;
    unsigned takesNull : 1;
} tableModifier;

// the modifier byte for MIN_MAX_MTYPE
typedef struct minMaxModifier
{
    unsigned size      : 2;
    unsigned sign      : 1;
    unsigned unused    : 3;
    unsigned hasBits   : 1;
    unsigned takesNull : 1;
} minMaxModifier;

// the modifier byte for ATTRIBUTES_MTYPE
typedef struct attributesModifier
{
    unsigned size   : 2;
    unsigned sign   : 1;
    unsigned unused : 5;
} attributesModifier;

// the modifier byte is not present in a STRUCTURE_MTYPE or an TPM2B_MTYPE

// the modifier byte for a TPM2BS_MTYPE
typedef struct tpm2bsModifier
{
    unsigned offset        : 4;
    unsigned unused        : 2;
    unsigned sizeEqual     : 1;
    unsigned propigateNull : 1;
} tpm2bsModifier;

// the modifier byte for a LIST_MTYPE
typedef struct listModifier
{
    unsigned offset        : 4;
    unsigned unused        : 2;
    unsigned sizeEqual     : 1;
    unsigned propigateNull : 1;
} listModifier;

//*** Modifier Octet Values
// These are in used in anything that is an integer value. Theses would not be in
// structure modifier bytes (they would be used in values in structures but not the
// STRUCTURE_MTYPE header.
#define ONE_BYTES   (0)
#define TWO_BYTES   (1)
#define FOUR_BYTES  (2)
#define EIGHT_BYTES (3)
#define SIZE_MASK   (0x3)
#define IS_SIGNED   (1 << 2)  // when the unmarshaled type is a signed value
#define SIGNED_MASK (SIZE_MASK | IS_SIGNED)

// This may be used for any type except a UINT_MTYPE
#define TAKES_NULL (1 << 7)  // when the type takes a null
```

```
// When referencing a structure, this flag indicates if a null is to be propagated
// to the referenced structure or type.
#define PROPAGATE_NULL (TAKES_NULL)

// Can be used in min-max or table structures.
#define HAS_BITS (1 << 6)  // when bit mask is present

// In a union, we need to know if this is a union of constant arrays.
#define IS_ARRAY_UNION (1 << 6)

// In a TPM2BS_MTYPE
#define SIZE_EQUAL  (1 << 6)
#define OFFSET_MASK (0xF)

// Right now, there are three spare bits in the modifiers field.

// Within the descriptor word of each entry in a StructMarsh_mst, there is a selector
// field to determine which of the sub-types the entry represents and a field that is
// used to reference another structure entry. This is a 6-bit field allowing a
// structure to have 64 entries. This should be more than enough as the structures are
// not that long. As of now, only 10-bits of the descriptor word leaving room for
// expansion.

// These are the values used in a STRUCTURE_MTYPE to identify the sub-type of the
// thing being processed
#define SIMPLE_STYPE 0
#define UNION_STYPE  1
#define ARRAY_STYPE  2

// The code used GET_ to get the element type and the compiler uses SET_ to initialize
// the value. The element type is the three bits (2:0).
#define GET_ELEMENT_TYPE(val) (val & 7)
#define SET_ELEMENT_TYPE(val) (val & 7)

// When an entry is an array or union, this references the structure entry that
// contains the dimension or selector value. The code then uses this number to look up
// the structure entry for that element to find out what it and where is it in memory.
// When this is not a reference, it is a simple type and it could be used as an array
// value or a union selector. When a simple value, this field contains the size
// of the associated value (ONE_BYTES, TWO_BYTES ...)
//
// The entry size/number is 6 bits (13:8).
#define GET_ELEMENT_NUMBER(val) (((val) >> 8) & 0x3F)
#define SET_ELEMENT_NUMBER(val) (((val) & 0x3F) << 8)
#define GET_ELEMENT_SIZE(val)   GET_ELEMENT_NUMBER(val)
#define SET_ELEMENT_SIZE(val)   SET_ELEMENT_NUMBER(val)
// This determines if the null flag is propagated to this type. If generate, the
// NULL_FLAG is SET in the index value. This flag is one bit (7)
#define ELEMENT_PROPAGATE (PROPAGATE_NULL)

#define INDEX_MASK ((UINT16)NULL_MASK)

// This is used in all bit-field checks. These are used when a value that is checked
// is conditional (dependent on the compilation). For example, if AES_128 is (NO),
// then the bit associated with AES_128 will be 0. In some cases, the bit value is
// found by checking that the input is within the range of the table, and then using
// the (val - min) value to index the bit. This would be used when verifying that
// a particular algorithm is implemented. In other cases, there is a bit for each
// value in a table. For example, if checking the key sizes, there is a list of
// possible key sizes allowed by the algorithm registry and a bit field to indicate
// if that key size is allowed in the implementation. The smallest bit field has
// 32-bits because it is implemented as part of the 'values' array in structures
// that allow bit fields.
#define IS_BIT_SET32(bit, bits) \
    ((((UINT32*)bits)[bit >> 5] & (1 << (bit & 0x1F))) != 0)
```

```
// For a COMPOSITE_MTYPE, the qualifiers byte has an element size and count.
#define SET_ELEMENT_COUNT(count) ((count & 0x1F) << 3)
#define GET_ELEMENT_COUNT(val)   ((val >> 3) & 0x1F)

#endif  // _TABLE_MARSHAL_H_
```

6.38 /tpm/include/private/TableMarshalDefines.h

```
#ifndef _TABLE_MARSHAL_DEFINES_H_
#define _TABLE_MARSHAL_DEFINES_H_

#define NULL_SHIFT 15
#define NULL_FLAG  (1 << NULL_SHIFT)

// The range macro processes a min, max value and produces a values that is used in
// the computation to see if something is within a range. The max value is (max-min).
// This lets the check for something ('val') within a range become:
//   if((val - min) <= max) // passes if in range
//   if((val - min) > max) // passes if not in range
// This works because all values are converted to UINT32 values before the compare.
// For (val - min), all values greater than or equal to val will become positive
// values with a value equal to 'min' being zero. This means that in an unsigned
// compare against 'max,' any value that is outside the range will appear to be a
// number greater than max. The benefit of this operation is that this will work even
// if the input value is a signed number as long as the input is sign extended.

#define RANGE(_min_, _max_, _base_) (UINT32) _min_, (UINT32)((_base_)(_max_ - _min_))

// This macro is like the offsetof macro but, instead of computing the offset of
// a structure element, it computes the stride between elements that are in a
// structure array. This is used instead of sizeof() because the sizeof() operator on
// a structure can return an implementation dependent value.
#define STRIDE(s) ((UINT16)(size_t) & (((s*)0)[1]))

#define MARSHAL_REF(TYPE) ((UINT16)(offsetof(MARSHAL_DATA, TYPE)))

// This macro creates the entry in the array lookup table
#define ARRAY_MARSHAL_ENTRY(TYPE)                                    \
    {                                                                \
        (marshalIndex_t) TYPE##_MARSHAL_REF, (UINT16)STRIDE(TYPE) \
    }

// Defines for array lookup
#define UINT8_ARRAY_MARSHAL_INDEX                  0   // 0x00
#define TPM_CC_ARRAY_MARSHAL_INDEX                 1   // 0x01
#define TPMA_CC_ARRAY_MARSHAL_INDEX                2   // 0x02
#define TPM_ALG_ID_ARRAY_MARSHAL_INDEX             3   // 0x03
#define TPM_HANDLE_ARRAY_MARSHAL_INDEX             4   // 0x04
#define TPM2B_DIGEST_ARRAY_MARSHAL_INDEX           5   // 0x05
#define TPMT_HA_ARRAY_MARSHAL_INDEX                6   // 0x06
#define TPMS_PCR_SELECTION_ARRAY_MARSHAL_INDEX     7   // 0x07
#define TPMS_ALG_PROPERTY_ARRAY_MARSHAL_INDEX      8   // 0x08
#define TPMS_TAGGED_PROPERTY_ARRAY_MARSHAL_INDEX   9   // 0x09
#define TPMS_TAGGED_PCR_SELECT_ARRAY_MARSHAL_INDEX 10  // 0x0A
#define TPM_ECC_CURVE_ARRAY_MARSHAL_INDEX          11  // 0x0B
#define TPMS_TAGGED_POLICY_ARRAY_MARSHAL_INDEX     12  // 0x0C
#define TPMS_ACT_DATA_ARRAY_MARSHAL_INDEX          13  // 0x0D
#define TPMS_AC_OUTPUT_ARRAY_MARSHAL_INDEX         14  // 0x0E

// Defines for referencing a type by offset
#define UINT8_MARSHAL_REF                  ((UINT16)(offsetof(MarshalData_st,
UINT8_DATA)))
#define BYTE_MARSHAL_REF                   UINT8_MARSHAL_REF
#define TPM_HT_MARSHAL_REF                 UINT8_MARSHAL_REF
#define TPMA_LOCALITY_MARSHAL_REF          UINT8_MARSHAL_REF
```

```
#define UINT16_MARSHAL_REF                  ((UINT16)(offsetof(MarshalData_st,
UINT16_DATA)))
#define TPM_KEY_SIZE_MARSHAL_REF            UINT16_MARSHAL_REF
#define TPM_KEY_BITS_MARSHAL_REF            UINT16_MARSHAL_REF
#define TPM_ALG_ID_MARSHAL_REF              UINT16_MARSHAL_REF
#define TPM_ST_MARSHAL_REF                  UINT16_MARSHAL_REF
#define UINT32_MARSHAL_REF                  ((UINT16)(offsetof(MarshalData_st,
UINT32_DATA)))
#define TPM_ALGORITHM_ID_MARSHAL_REF        UINT32_MARSHAL_REF
#define TPM_MODIFIER_INDICATOR_MARSHAL_REF UINT32_MARSHAL_REF
#define TPM_AUTHORIZATION_SIZE_MARSHAL_REF UINT32_MARSHAL_REF
#define TPM_PARAMETER_SIZE_MARSHAL_REF      UINT32_MARSHAL_REF
#define TPM_SPEC_MARSHAL_REF                UINT32_MARSHAL_REF
#define TPM_CONSTANTS32_MARSHAL_REF         UINT32_MARSHAL_REF
#define TPM_CC_MARSHAL_REF                  UINT32_MARSHAL_REF
#define TPM_RC_MARSHAL_REF                  UINT32_MARSHAL_REF
#define TPM_PT_MARSHAL_REF                  UINT32_MARSHAL_REF
#define TPM_PT_PCR_MARSHAL_REF              UINT32_MARSHAL_REF
#define TPM_PS_MARSHAL_REF                  UINT32_MARSHAL_REF
#define TPM_HANDLE_MARSHAL_REF              UINT32_MARSHAL_REF
#define TPM_RH_MARSHAL_REF                  UINT32_MARSHAL_REF
#define TPM_HC_MARSHAL_REF                  UINT32_MARSHAL_REF
#define TPMA_PERMANENT_MARSHAL_REF          UINT32_MARSHAL_REF
#define TPMA_STARTUP_CLEAR_MARSHAL_REF      UINT32_MARSHAL_REF
#define TPMA_MEMORY_MARSHAL_REF             UINT32_MARSHAL_REF
#define TPMA_CC_MARSHAL_REF                 UINT32_MARSHAL_REF
#define TPMA_MODES_MARSHAL_REF              UINT32_MARSHAL_REF
#define TPMA_X509_KEY_USAGE_MARSHAL_REF     UINT32_MARSHAL_REF
#define TPM_NV_INDEX_MARSHAL_REF            UINT32_MARSHAL_REF
#define TPM_AE_MARSHAL_REF                  UINT32_MARSHAL_REF
#define UINT64_MARSHAL_REF                  ((UINT16)(offsetof(MarshalData_st,
UINT64_DATA)))
#define INT8_MARSHAL_REF                    ((UINT16)(offsetof(MarshalData_st,
INT8_DATA)))
#define INT16_MARSHAL_REF                   ((UINT16)(offsetof(MarshalData_st,
INT16_DATA)))
#define INT32_MARSHAL_REF                   ((UINT16)(offsetof(MarshalData_st,
INT32_DATA)))
#define INT64_MARSHAL_REF                   ((UINT16)(offsetof(MarshalData_st,
INT64_DATA)))
#define UINT0_MARSHAL_REF                   ((UINT16)(offsetof(MarshalData_st,
UINT0_DATA)))
#define TPM_ECC_CURVE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM_ECC_CURVE_DATA)))
#define TPM_CLOCK_ADJUST_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM_CLOCK_ADJUST_DATA)))
#define TPM_EO_MARSHAL_REF  ((UINT16)(offsetof(MarshalData_st, TPM_EO_DATA)))
#define TPM_SU_MARSHAL_REF  ((UINT16)(offsetof(MarshalData_st, TPM_SU_DATA)))
#define TPM_SE_MARSHAL_REF  ((UINT16)(offsetof(MarshalData_st, TPM_SE_DATA)))
#define TPM_CAP_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPM_CAP_DATA)))
#define TPMA_ALGORITHM_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMA_ALGORITHM_DATA)))
#define TPMA_OBJECT_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMA_OBJECT_DATA)))
#define TPMA_SESSION_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMA_SESSION_DATA)))
#define TPMA_ACT_MARSHAL_REF    ((UINT16)(offsetof(MarshalData_st, TPMA_ACT_DATA)))
#define TPMI_YES_NO_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMI_YES_NO_DATA)))
#define TPMI_DH_OBJECT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_DH_OBJECT_DATA)))
#define TPMI_DH_PARENT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_DH_PARENT_DATA)))
#define TPMI_DH_PERSISTENT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_DH_PERSISTENT_DATA)))
#define TPMI_DH_ENTITY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_DH_ENTITY_DATA)))
#define TPMI_DH_PCR_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMI_DH_PCR_DATA)))
```

```
#define TPMI_SH_AUTH_SESSION_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_SH_AUTH_SESSION_DATA)))
#define TPMI_SH_HMAC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_SH_HMAC_DATA)))
#define TPMI_SH_POLICY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_SH_POLICY_DATA)))
#define TPMI_DH_CONTEXT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_DH_CONTEXT_DATA)))
#define TPMI_DH_SAVED_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_DH_SAVED_DATA)))
#define TPMI_RH_HIERARCHY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_HIERARCHY_DATA)))
#define TPMI_RH_ENABLES_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_ENABLES_DATA)))
#define TPMI_RH_HIERARCHY_AUTH_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_HIERARCHY_AUTH_DATA)))
#define TPMI_RH_HIERARCHY_POLICY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_HIERARCHY_POLICY_DATA)))
#define TPMI_RH_BASE_HIERARCHY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_BASE_HIERARCHY_DATA)))
#define TPMI_RH_PLATFORM_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_PLATFORM_DATA)))
#define TPMI_RH_OWNER_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_OWNER_DATA)))
#define TPMI_RH_ENDORSEMENT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_ENDORSEMENT_DATA)))
#define TPMI_RH_PROVISION_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_PROVISION_DATA)))
#define TPMI_RH_CLEAR_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_CLEAR_DATA)))
#define TPMI_RH_NV_AUTH_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_NV_AUTH_DATA)))
#define TPMI_RH_LOCKOUT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_LOCKOUT_DATA)))
#define TPMI_RH_NV_INDEX_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_NV_INDEX_DATA)))
#define TPMI_RH_NV_DEFINED_INDEX_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_NV_DEFINED_INDEX_DATA)))
#define TPMI_RH_NV_LEGACY_INDEX_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_NV_LEGACY_INDEX_DATA)))
#define TPMI_RH_NV_EXP_INDEX_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RH_NV_EXP_INDEX_DATA)))
#define TPMI_RH_AC_MARSHAL_REF  ((UINT16)(offsetof(MarshalData_st, TPMI_RH_AC_DATA)))
#define TPMI_RH_ACT_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMI_RH_ACT_DATA)))
#define TPMI_ALG_HASH_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_HASH_DATA)))
#define TPMI_ALG_ASYM_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_ASYM_DATA)))
#define TPMI_ALG_SYM_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_SYM_DATA)))
#define TPMI_ALG_SYM_OBJECT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_SYM_OBJECT_DATA)))
#define TPMI_ALG_SYM_MODE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_SYM_MODE_DATA)))
#define TPMI_ALG_KDF_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_KDF_DATA)))
#define TPMI_ALG_SIG_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_SIG_SCHEME_DATA)))
#define TPMI_ECC_KEY_EXCHANGE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ECC_KEY_EXCHANGE_DATA)))
#define TPMI_ST_COMMAND_TAG_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ST_COMMAND_TAG_DATA)))
#define TPMI_ALG_MAC_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_MAC_SCHEME_DATA)))
#define TPMI_ALG_CIPHER_MODE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_CIPHER_MODE_DATA)))
```

```
#define TPMS_EMPTY_MARSHAL_REF            ((UINT16)(offsetof(MarshalData_st,
TPMS_EMPTY_DATA)))
#define TPMS_ENC_SCHEME_RSAES_MARSHAL_REF TPMS_EMPTY_MARSHAL_REF
#define TPMS_ALGORITHM_DESCRIPTION_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_ALGORITHM_DESCRIPTION_DATA)))
#define TPMU_HA_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMU_HA_DATA)))
#define TPMT_HA_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMT_HA_DATA)))
#define TPM2B_DIGEST_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_DIGEST_DATA)))
#define TPM2B_NONCE_MARSHAL_REF   TPM2B_DIGEST_MARSHAL_REF
#define TPM2B_AUTH_MARSHAL_REF    TPM2B_DIGEST_MARSHAL_REF
#define TPM2B_OPERAND_MARSHAL_REF TPM2B_DIGEST_MARSHAL_REF
#define TPM2B_DATA_MARSHAL_REF    ((UINT16)(offsetof(MarshalData_st,
TPM2B_DATA_DATA)))
#define TPM2B_EVENT_MARSHAL_REF   ((UINT16)(offsetof(MarshalData_st,
TPM2B_EVENT_DATA)))
#define TPM2B_MAX_BUFFER_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_MAX_BUFFER_DATA)))
#define TPM2B_MAX_NV_BUFFER_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_MAX_NV_BUFFER_DATA)))
#define TPM2B_TIMEOUT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_TIMEOUT_DATA)))
#define TPM2B_IV_MARSHAL_REF              ((UINT16)(offsetof(MarshalData_st,
TPM2B_IV_DATA)))
#define NULL_UNION_MARSHAL_REF            ((UINT16)(offsetof(MarshalData_st,
NULL_UNION_DATA)))
#define TPMU_NAME_MARSHAL_REF             NULL_UNION_MARSHAL_REF
#define TPMU_SENSITIVE_CREATE_MARSHAL_REF NULL_UNION_MARSHAL_REF
#define TPM2B_NAME_MARSHAL_REF            ((UINT16)(offsetof(MarshalData_st,
TPM2B_NAME_DATA)))
#define TPMS_PCR_SELECT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_PCR_SELECT_DATA)))
#define TPMS_PCR_SELECTION_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_PCR_SELECTION_DATA)))
#define TPMT_TK_CREATION_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_TK_CREATION_DATA)))
#define TPMT_TK_VERIFIED_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_TK_VERIFIED_DATA)))
#define TPMT_TK_AUTH_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_TK_AUTH_DATA)))
#define TPMT_TK_HASHCHECK_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_TK_HASHCHECK_DATA)))
#define TPMS_ALG_PROPERTY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_ALG_PROPERTY_DATA)))
#define TPMS_TAGGED_PROPERTY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_TAGGED_PROPERTY_DATA)))
#define TPMS_TAGGED_PCR_SELECT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_TAGGED_PCR_SELECT_DATA)))
#define TPMS_TAGGED_POLICY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_TAGGED_POLICY_DATA)))
#define TPMS_ACT_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_ACT_DATA_DATA)))
#define TPML_CC_MARSHAL_REF     ((UINT16)(offsetof(MarshalData_st, TPML_CC_DATA)))
#define TPML_CCA_MARSHAL_REF    ((UINT16)(offsetof(MarshalData_st, TPML_CCA_DATA)))
#define TPML_ALG_MARSHAL_REF    ((UINT16)(offsetof(MarshalData_st, TPML_ALG_DATA)))
#define TPML_HANDLE_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPML_HANDLE_DATA)))
#define TPML_DIGEST_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPML_DIGEST_DATA)))
#define TPML_DIGEST_VALUES_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_DIGEST_VALUES_DATA)))
#define TPML_PCR_SELECTION_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_PCR_SELECTION_DATA)))
#define TPML_ALG_PROPERTY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_ALG_PROPERTY_DATA)))
#define TPML_TAGGED_TPM_PROPERTY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_TAGGED_TPM_PROPERTY_DATA)))
#define TPML_TAGGED_PCR_PROPERTY_MARSHAL_REF \
```

```
    ((UINT16)(offsetof(MarshalData_st, TPML_TAGGED_PCR_PROPERTY_DATA)))
#define TPML_ECC_CURVE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_ECC_CURVE_DATA)))
#define TPML_TAGGED_POLICY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_TAGGED_POLICY_DATA)))
#define TPML_ACT_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_ACT_DATA_DATA)))
#define TPMU_CAPABILITIES_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_CAPABILITIES_DATA)))
#define TPMS_CAPABILITY_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_CAPABILITY_DATA_DATA)))
#define TPMU_SET_CAPABILITIES_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_SET_CAPABILITIES_DATA)))
#define TPMS_SET_CAPABILITY_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SET_CAPABILITY_DATA_DATA)))
#define TPM2B_SET_CAPABILITY_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_SET_CAPABILITY_DATA_DATA)))
#define TPMS_CLOCK_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_CLOCK_INFO_DATA)))
#define TPMS_TIME_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_TIME_INFO_DATA)))
#define TPMS_TIME_ATTEST_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_TIME_ATTEST_INFO_DATA)))
#define TPMS_CERTIFY_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_CERTIFY_INFO_DATA)))
#define TPMS_QUOTE_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_QUOTE_INFO_DATA)))
#define TPMS_COMMAND_AUDIT_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_COMMAND_AUDIT_INFO_DATA)))
#define TPMS_SESSION_AUDIT_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SESSION_AUDIT_INFO_DATA)))
#define TPMS_CREATION_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_CREATION_INFO_DATA)))
#define TPMS_NV_CERTIFY_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_NV_CERTIFY_INFO_DATA)))
#define TPMS_NV_DIGEST_CERTIFY_INFO_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_NV_DIGEST_CERTIFY_INFO_DATA)))
#define TPMI_ST_ATTEST_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ST_ATTEST_DATA)))
#define TPMU_ATTEST_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMU_ATTEST_DATA)))
#define TPMS_ATTEST_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMS_ATTEST_DATA)))
#define TPM2B_ATTEST_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_ATTEST_DATA)))
#define TPMS_AUTH_COMMAND_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_AUTH_COMMAND_DATA)))
#define TPMS_AUTH_RESPONSE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_AUTH_RESPONSE_DATA)))
#define TPMI_TDES_KEY_BITS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_TDES_KEY_BITS_DATA)))
#define TPMI_AES_KEY_BITS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_AES_KEY_BITS_DATA)))
#define TPMI_SM4_KEY_BITS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_SM4_KEY_BITS_DATA)))
#define TPMI_CAMELLIA_KEY_BITS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_CAMELLIA_KEY_BITS_DATA)))
#define TPMU_SYM_KEY_BITS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_SYM_KEY_BITS_DATA)))
#define TPMU_SYM_MODE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_SYM_MODE_DATA)))
#define TPMT_SYM_DEF_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_SYM_DEF_DATA)))
#define TPMT_SYM_DEF_OBJECT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_SYM_DEF_OBJECT_DATA)))
#define TPM2B_SYM_KEY_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_SYM_KEY_DATA)))
#define TPMS_SYMCIPHER_PARMS_MARSHAL_REF \
```

```
    ((UINT16)(offsetof(MarshalData_st, TPMS_SYMCIPHER_PARMS_DATA)))
#define TPM2B_LABEL_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPM2B_LABEL_DATA)))
#define TPMS_DERIVE_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMS_DERIVE_DATA)))
#define TPM2B_DERIVE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_DERIVE_DATA)))
#define TPM2B_SENSITIVE_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_SENSITIVE_DATA_DATA)))
#define TPMS_SENSITIVE_CREATE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SENSITIVE_CREATE_DATA)))
#define TPM2B_SENSITIVE_CREATE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_SENSITIVE_CREATE_DATA)))
#define TPMS_SCHEME_HASH_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SCHEME_HASH_DATA)))
#define TPMS_SCHEME_HMAC_MARSHAL_REF                TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_SIG_SCHEME_RSASSA_MARSHAL_REF          TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_SIG_SCHEME_RSAPSS_MARSHAL_REF          TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_SIG_SCHEME_ECDSA_MARSHAL_REF           TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_SIG_SCHEME_SM2_MARSHAL_REF             TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_SIG_SCHEME_ECSCHNORR_MARSHAL_REF       TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_ENC_SCHEME_OAEP_MARSHAL_REF            TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_KEY_SCHEME_ECDH_MARSHAL_REF            TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_KEY_SCHEME_ECMQV_MARSHAL_REF           TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_KDF_SCHEME_MGF1_MARSHAL_REF            TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_KDF_SCHEME_KDF1_SP800_56A_MARSHAL_REF TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_KDF_SCHEME_KDF2_MARSHAL_REF            TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_KDF_SCHEME_KDF1_SP800_108_MARSHAL_REF TPMS_SCHEME_HASH_MARSHAL_REF
#define TPMS_SCHEME_ECDAA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SCHEME_ECDAA_DATA)))
#define TPMS_SIG_SCHEME_ECDAA_MARSHAL_REF TPMS_SCHEME_ECDAA_MARSHAL_REF
#define TPMI_ALG_KEYEDHASH_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_KEYEDHASH_SCHEME_DATA)))
#define TPMS_SCHEME_XOR_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SCHEME_XOR_DATA)))
#define TPMU_SCHEME_KEYEDHASH_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_SCHEME_KEYEDHASH_DATA)))
#define TPMT_KEYEDHASH_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_KEYEDHASH_SCHEME_DATA)))
#define TPMU_SIG_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_SIG_SCHEME_DATA)))
#define TPMT_SIG_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_SIG_SCHEME_DATA)))
#define TPMU_KDF_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_KDF_SCHEME_DATA)))
#define TPMT_KDF_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_KDF_SCHEME_DATA)))
#define TPMI_ALG_ASYM_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_ASYM_SCHEME_DATA)))
#define TPMU_ASYM_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_ASYM_SCHEME_DATA)))
#define TPMI_ALG_RSA_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_RSA_SCHEME_DATA)))
#define TPMT_RSA_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_RSA_SCHEME_DATA)))
#define TPMI_ALG_RSA_DECRYPT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_RSA_DECRYPT_DATA)))
#define TPMT_RSA_DECRYPT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_RSA_DECRYPT_DATA)))
#define TPM2B_PUBLIC_KEY_RSA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_PUBLIC_KEY_RSA_DATA)))
#define TPMI_RSA_KEY_BITS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_RSA_KEY_BITS_DATA)))
#define TPM2B_PRIVATE_KEY_RSA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_PRIVATE_KEY_RSA_DATA)))
#define TPM2B_ECC_PARAMETER_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_ECC_PARAMETER_DATA)))
#define TPMS_ECC_POINT_MARSHAL_REF \
```

```
    ((UINT16)(offsetof(MarshalData_st, TPMS_ECC_POINT_DATA)))
#define TPM2B_ECC_POINT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_ECC_POINT_DATA)))
#define TPMI_ALG_ECC_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_ECC_SCHEME_DATA)))
#define TPMI_ECC_CURVE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ECC_CURVE_DATA)))
#define TPMT_ECC_SCHEME_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_ECC_SCHEME_DATA)))
#define TPMS_ALGORITHM_DETAIL_ECC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_ALGORITHM_DETAIL_ECC_DATA)))
#define TPMS_SIGNATURE_RSA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SIGNATURE_RSA_DATA)))
#define TPMS_SIGNATURE_RSASSA_MARSHAL_REF TPMS_SIGNATURE_RSA_MARSHAL_REF
#define TPMS_SIGNATURE_RSAPSS_MARSHAL_REF TPMS_SIGNATURE_RSA_MARSHAL_REF
#define TPMS_SIGNATURE_ECC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_SIGNATURE_ECC_DATA)))
#define TPMS_SIGNATURE_ECDAA_MARSHAL_REF     TPMS_SIGNATURE_ECC_MARSHAL_REF
#define TPMS_SIGNATURE_ECDSA_MARSHAL_REF     TPMS_SIGNATURE_ECC_MARSHAL_REF
#define TPMS_SIGNATURE_SM2_MARSHAL_REF       TPMS_SIGNATURE_ECC_MARSHAL_REF
#define TPMS_SIGNATURE_ECSCHNORR_MARSHAL_REF TPMS_SIGNATURE_ECC_MARSHAL_REF
#define TPMU_SIGNATURE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_SIGNATURE_DATA)))
#define TPMT_SIGNATURE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_SIGNATURE_DATA)))
#define TPMU_ENCRYPTED_SECRET_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_ENCRYPTED_SECRET_DATA)))
#define TPM2B_ENCRYPTED_SECRET_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_ENCRYPTED_SECRET_DATA)))
#define TPMI_ALG_PUBLIC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMI_ALG_PUBLIC_DATA)))
#define TPMU_PUBLIC_ID_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_PUBLIC_ID_DATA)))
#define TPMS_KEYEDHASH_PARMS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_KEYEDHASH_PARMS_DATA)))
#define TPMS_RSA_PARMS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_RSA_PARMS_DATA)))
#define TPMS_ECC_PARMS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_ECC_PARMS_DATA)))
#define TPMU_PUBLIC_PARMS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_PUBLIC_PARMS_DATA)))
#define TPMT_PUBLIC_PARMS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_PUBLIC_PARMS_DATA)))
#define TPMT_PUBLIC_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMT_PUBLIC_DATA)))
#define TPM2B_PUBLIC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_PUBLIC_DATA)))
#define TPM2B_TEMPLATE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_TEMPLATE_DATA)))
#define TPM2B_PRIVATE_VENDOR_SPECIFIC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_PRIVATE_VENDOR_SPECIFIC_DATA)))
#define TPMU_SENSITIVE_COMPOSITE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_SENSITIVE_COMPOSITE_DATA)))
#define TPMT_SENSITIVE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_SENSITIVE_DATA)))
#define TPM2B_SENSITIVE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_SENSITIVE_DATA)))
#define TPM2B_PRIVATE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_PRIVATE_DATA)))
#define TPM2B_ID_OBJECT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_ID_OBJECT_DATA)))
#define TPMS_NV_PIN_COUNTER_PARAMETERS_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_NV_PIN_COUNTER_PARAMETERS_DATA)))
#define TPMA_NV_MARSHAL_REF     ((UINT16)(offsetof(MarshalData_st, TPMA_NV_DATA)))
#define TPMA_NV_EXP_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPMA_NV_EXP_DATA)))
#define TPMS_NV_PUBLIC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_NV_PUBLIC_DATA)))
```

```
#define TPM2B_NV_PUBLIC_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_NV_PUBLIC_DATA)))
#define TPMS_NV_PUBLIC_EXP_ATTR_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_NV_PUBLIC_EXP_ATTR_DATA)))
#define TPMU_NV_PUBLIC_2_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMU_NV_PUBLIC_2_DATA)))
#define TPMT_NV_PUBLIC_2_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMT_NV_PUBLIC_2_DATA)))
#define TPM2B_NV_PUBLIC_2_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_NV_PUBLIC_2_DATA)))
#define TPM2B_CONTEXT_SENSITIVE_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_CONTEXT_SENSITIVE_DATA)))
#define TPMS_CONTEXT_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_CONTEXT_DATA_DATA)))
#define TPM2B_CONTEXT_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_CONTEXT_DATA_DATA)))
#define TPMS_CONTEXT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_CONTEXT_DATA)))
#define TPMS_CREATION_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_CREATION_DATA_DATA)))
#define TPM2B_CREATION_DATA_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPM2B_CREATION_DATA_DATA)))
#define TPM_AT_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, TPM_AT_DATA)))
#define TPMS_AC_OUTPUT_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPMS_AC_OUTPUT_DATA)))
#define TPML_AC_CAPABILITIES_MARSHAL_REF \
    ((UINT16)(offsetof(MarshalData_st, TPML_AC_CAPABILITIES_DATA)))
#define Type00_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type00_DATA)))
#define Type01_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type01_DATA)))
#define Type02_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type02_DATA)))
#define Type03_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type03_DATA)))
#define Type04_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type04_DATA)))
#define Type05_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type05_DATA)))
#define Type06_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type06_DATA)))
#define Type07_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type07_DATA)))
#define Type08_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type08_DATA)))
#define Type09_MARSHAL_REF Type08_MARSHAL_REF
#define Type14_MARSHAL_REF Type08_MARSHAL_REF
#define Type10_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type10_DATA)))
#define Type11_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type11_DATA)))
#define Type12_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type12_DATA)))
#define Type13_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type13_DATA)))
#define Type15_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type15_DATA)))
#define Type16_MARSHAL_REF Type15_MARSHAL_REF
#define Type17_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type17_DATA)))
#define Type18_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type18_DATA)))
#define Type19_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type19_DATA)))
#define Type20_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type20_DATA)))
#define Type21_MARSHAL_REF Type20_MARSHAL_REF
#define Type22_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type22_DATA)))
#define Type23_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type23_DATA)))
#define Type24_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type24_DATA)))
#define Type25_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type25_DATA)))
#define Type26_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type26_DATA)))
#define Type27_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type27_DATA)))
#define Type28_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type28_DATA)))
#define Type29_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type29_DATA)))
#define Type30_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type30_DATA)))
#define Type31_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type31_DATA)))
#define Type32_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type32_DATA)))
#define Type33_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type33_DATA)))
#define Type34_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type34_DATA)))
#define Type35_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type35_DATA)))
#define Type36_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type36_DATA)))
#define Type37_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type37_DATA)))
#define Type38_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type38_DATA)))
```

```
#define Type39_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type39_DATA)))
#define Type40_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type40_DATA)))
#define Type41_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type41_DATA)))
#define Type42_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type42_DATA)))
#define Type43_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type43_DATA)))
#define Type44_MARSHAL_REF ((UINT16)(offsetof(MarshalData_st, Type44_DATA)))

//#defines to change calling sequence for code using marshaling
#define UINT8_Unmarshal(target, buffer, size) \
    Unmarshal(UINT8_MARSHAL_REF, (target), (buffer), (size))
#define UINT8_Marshal(source, buffer, size) \
    Marshal(UINT8_MARSHAL_REF, (source), (buffer), (size))
#define BYTE_Unmarshal(target, buffer, size) \
    Unmarshal(UINT8_MARSHAL_REF, (target), (buffer), (size))
#define BYTE_Marshal(source, buffer, size) \
    Marshal(UINT8_MARSHAL_REF, (source), (buffer), (size))
#define INT8_Unmarshal(target, buffer, size) \
    Unmarshal(INT8_MARSHAL_REF, (target), (buffer), (size))
#define INT8_Marshal(source, buffer, size) \
    Marshal(INT8_MARSHAL_REF, (source), (buffer), (size))
#define UINT16_Unmarshal(target, buffer, size) \
    Unmarshal(UINT16_MARSHAL_REF, (target), (buffer), (size))
#define UINT16_Marshal(source, buffer, size) \
    Marshal(UINT16_MARSHAL_REF, (source), (buffer), (size))
#define INT16_Unmarshal(target, buffer, size) \
    Unmarshal(INT16_MARSHAL_REF, (target), (buffer), (size))
#define INT16_Marshal(source, buffer, size) \
    Marshal(INT16_MARSHAL_REF, (source), (buffer), (size))
#define UINT32_Unmarshal(target, buffer, size) \
    Unmarshal(UINT32_MARSHAL_REF, (target), (buffer), (size))
#define UINT32_Marshal(source, buffer, size) \
    Marshal(UINT32_MARSHAL_REF, (source), (buffer), (size))
#define INT32_Unmarshal(target, buffer, size) \
    Unmarshal(INT32_MARSHAL_REF, (target), (buffer), (size))
#define INT32_Marshal(source, buffer, size) \
    Marshal(INT32_MARSHAL_REF, (source), (buffer), (size))
#define UINT64_Unmarshal(target, buffer, size) \
    Unmarshal(UINT64_MARSHAL_REF, (target), (buffer), (size))
#define UINT64_Marshal(source, buffer, size) \
    Marshal(UINT64_MARSHAL_REF, (source), (buffer), (size))
#define INT64_Unmarshal(target, buffer, size) \
    Unmarshal(INT64_MARSHAL_REF, (target), (buffer), (size))
#define INT64_Marshal(source, buffer, size) \
    Marshal(INT64_MARSHAL_REF, (source), (buffer), (size))
#define TPM_ALGORITHM_ID_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_ALGORITHM_ID_MARSHAL_REF, (target), (buffer), (size))
#define TPM_ALGORITHM_ID_Marshal(source, buffer, size) \
    Marshal(TPM_ALGORITHM_ID_MARSHAL_REF, (source), (buffer), (size))
#define TPM_MODIFIER_INDICATOR_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_MODIFIER_INDICATOR_MARSHAL_REF, (target), (buffer), (size))
#define TPM_MODIFIER_INDICATOR_Marshal(source, buffer, size) \
    Marshal(TPM_MODIFIER_INDICATOR_MARSHAL_REF, (source), (buffer), (size))
#define TPM_AUTHORIZATION_SIZE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_AUTHORIZATION_SIZE_MARSHAL_REF, (target), (buffer), (size))
#define TPM_AUTHORIZATION_SIZE_Marshal(source, buffer, size) \
    Marshal(TPM_AUTHORIZATION_SIZE_MARSHAL_REF, (source), (buffer), (size))
#define TPM_PARAMETER_SIZE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_PARAMETER_SIZE_MARSHAL_REF, (target), (buffer), (size))
#define TPM_PARAMETER_SIZE_Marshal(source, buffer, size) \
    Marshal(TPM_PARAMETER_SIZE_MARSHAL_REF, (source), (buffer), (size))
#define TPM_KEY_SIZE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_KEY_SIZE_MARSHAL_REF, (target), (buffer), (size))
#define TPM_KEY_SIZE_Marshal(source, buffer, size) \
    Marshal(TPM_KEY_SIZE_MARSHAL_REF, (source), (buffer), (size))
#define TPM_KEY_BITS_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_KEY_BITS_MARSHAL_REF, (target), (buffer), (size))
```

```
#define TPM_KEY_BITS_Marshal(source, buffer, size) \
    Marshal(TPM_KEY_BITS_MARSHAL_REF, (source), (buffer), (size))
#define TPM_CONSTANTS32_Marshal(source, buffer, size) \
    Marshal(TPM_CONSTANTS32_MARSHAL_REF, (source), (buffer), (size))
#define TPM_ALG_ID_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_ALG_ID_MARSHAL_REF, (target), (buffer), (size))
#define TPM_ALG_ID_Marshal(source, buffer, size) \
    Marshal(TPM_ALG_ID_MARSHAL_REF, (source), (buffer), (size))
#define TPM_ECC_CURVE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_ECC_CURVE_MARSHAL_REF, (target), (buffer), (size))
#define TPM_ECC_CURVE_Marshal(source, buffer, size) \
    Marshal(TPM_ECC_CURVE_MARSHAL_REF, (source), (buffer), (size))
#define TPM_CC_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_CC_MARSHAL_REF, (target), (buffer), (size))
#define TPM_CC_Marshal(source, buffer, size) \
    Marshal(TPM_CC_MARSHAL_REF, (source), (buffer), (size))
#define TPM_RC_Marshal(source, buffer, size) \
    Marshal(TPM_RC_MARSHAL_REF, (source), (buffer), (size))
#define TPM_CLOCK_ADJUST_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_CLOCK_ADJUST_MARSHAL_REF, (target), (buffer), (size))
#define TPM_EO_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_EO_MARSHAL_REF, (target), (buffer), (size))
#define TPM_EO_Marshal(source, buffer, size) \
    Marshal(TPM_EO_MARSHAL_REF, (source), (buffer), (size))
#define TPM_ST_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_ST_MARSHAL_REF, (target), (buffer), (size))
#define TPM_ST_Marshal(source, buffer, size) \
    Marshal(TPM_ST_MARSHAL_REF, (source), (buffer), (size))
#define TPM_SU_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_SU_MARSHAL_REF, (target), (buffer), (size))
#define TPM_SE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_SE_MARSHAL_REF, (target), (buffer), (size))
#define TPM_CAP_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_CAP_MARSHAL_REF, (target), (buffer), (size))
#define TPM_CAP_Marshal(source, buffer, size) \
    Marshal(TPM_CAP_MARSHAL_REF, (source), (buffer), (size))
#define TPM_PT_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_PT_MARSHAL_REF, (target), (buffer), (size))
#define TPM_PT_Marshal(source, buffer, size) \
    Marshal(TPM_PT_MARSHAL_REF, (source), (buffer), (size))
#define TPM_PT_PCR_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_PT_PCR_MARSHAL_REF, (target), (buffer), (size))
#define TPM_PT_PCR_Marshal(source, buffer, size) \
    Marshal(TPM_PT_PCR_MARSHAL_REF, (source), (buffer), (size))
#define TPM_PS_Marshal(source, buffer, size) \
    Marshal(TPM_PS_MARSHAL_REF, (source), (buffer), (size))
#define TPM_HANDLE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_HANDLE_MARSHAL_REF, (target), (buffer), (size))
#define TPM_HANDLE_Marshal(source, buffer, size) \
    Marshal(TPM_HANDLE_MARSHAL_REF, (source), (buffer), (size))
#define TPM_HT_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_HT_MARSHAL_REF, (target), (buffer), (size))
#define TPM_HT_Marshal(source, buffer, size) \
    Marshal(TPM_HT_MARSHAL_REF, (source), (buffer), (size))
#define TPM_RH_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_RH_MARSHAL_REF, (target), (buffer), (size))
#define TPM_RH_Marshal(source, buffer, size) \
    Marshal(TPM_RH_MARSHAL_REF, (source), (buffer), (size))
#define TPM_HC_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_HC_MARSHAL_REF, (target), (buffer), (size))
#define TPM_HC_Marshal(source, buffer, size) \
    Marshal(TPM_HC_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_ALGORITHM_Unmarshal(target, buffer, size) \
    Unmarshal(TPMA_ALGORITHM_MARSHAL_REF, (target), (buffer), (size))
#define TPMA_ALGORITHM_Marshal(source, buffer, size) \
    Marshal(TPMA_ALGORITHM_MARSHAL_REF, (source), (buffer), (size))
```

```
#define TPMA_OBJECT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMA_OBJECT_MARSHAL_REF, (target), (buffer), (size))
#define TPMA_OBJECT_Marshal(source, buffer, size) \
    Marshal(TPMA_OBJECT_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_SESSION_Unmarshal(target, buffer, size) \
    Unmarshal(TPMA_SESSION_MARSHAL_REF, (target), (buffer), (size))
#define TPMA_SESSION_Marshal(source, buffer, size) \
    Marshal(TPMA_SESSION_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_LOCALITY_Unmarshal(target, buffer, size) \
    Unmarshal(TPMA_LOCALITY_MARSHAL_REF, (target), (buffer), (size))
#define TPMA_LOCALITY_Marshal(source, buffer, size) \
    Marshal(TPMA_LOCALITY_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_PERMANENT_Marshal(source, buffer, size) \
    Marshal(TPMA_PERMANENT_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_STARTUP_CLEAR_Marshal(source, buffer, size) \
    Marshal(TPMA_STARTUP_CLEAR_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_MEMORY_Marshal(source, buffer, size) \
    Marshal(TPMA_MEMORY_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_CC_Marshal(source, buffer, size) \
    Marshal(TPMA_CC_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_MODES_Marshal(source, buffer, size) \
    Marshal(TPMA_MODES_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_X509_KEY_USAGE_Marshal(source, buffer, size) \
    Marshal(TPMA_X509_KEY_USAGE_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_ACT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMA_ACT_MARSHAL_REF, (target), (buffer), (size))
#define TPMA_ACT_Marshal(source, buffer, size) \
    Marshal(TPMA_ACT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_YES_NO_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_YES_NO_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_YES_NO_Marshal(source, buffer, size) \
    Marshal(TPMI_YES_NO_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_DH_OBJECT_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_DH_OBJECT_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                            \
              (buffer),                                            \
              (size))
#define TPMI_DH_OBJECT_Marshal(source, buffer, size) \
    Marshal(TPMI_DH_OBJECT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_DH_PARENT_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_DH_PARENT_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                            \
              (buffer),                                            \
              (size))
#define TPMI_DH_PARENT_Marshal(source, buffer, size) \
    Marshal(TPMI_DH_PARENT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_DH_PERSISTENT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_DH_PERSISTENT_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_DH_PERSISTENT_Marshal(source, buffer, size) \
    Marshal(TPMI_DH_PERSISTENT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_DH_ENTITY_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_DH_ENTITY_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                            \
              (buffer),                                            \
              (size))
#define TPMI_DH_PCR_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_DH_PCR_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                         \
              (buffer),                                         \
              (size))
#define TPMI_SH_AUTH_SESSION_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_SH_AUTH_SESSION_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                  \
              (buffer),                                                  \
              (size))
#define TPMI_SH_AUTH_SESSION_Marshal(source, buffer, size) \
```

```
    Marshal(TPMI_SH_AUTH_SESSION_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_SH_HMAC_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_SH_HMAC_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_SH_HMAC_Marshal(source, buffer, size) \
    Marshal(TPMI_SH_HMAC_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_SH_POLICY_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_SH_POLICY_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_SH_POLICY_Marshal(source, buffer, size) \
    Marshal(TPMI_SH_POLICY_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_DH_CONTEXT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_DH_CONTEXT_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_DH_CONTEXT_Marshal(source, buffer, size) \
    Marshal(TPMI_DH_CONTEXT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_DH_SAVED_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_DH_SAVED_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_DH_SAVED_Marshal(source, buffer, size) \
    Marshal(TPMI_DH_SAVED_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_RH_HIERARCHY_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_RH_HIERARCHY_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                               \
              (buffer),                                               \
              (size))
#define TPMI_RH_HIERARCHY_Marshal(source, buffer, size) \
    Marshal(TPMI_RH_HIERARCHY_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_RH_ENABLES_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_RH_ENABLES_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                             \
              (buffer),                                             \
              (size))
#define TPMI_RH_ENABLES_Marshal(source, buffer, size) \
    Marshal(TPMI_RH_ENABLES_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_RH_HIERARCHY_AUTH_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_HIERARCHY_AUTH_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_HIERARCHY_POLICY_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_HIERARCHY_POLICY_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_PLATFORM_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_PLATFORM_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_OWNER_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_RH_OWNER_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                           \
              (buffer),                                           \
              (size))
#define TPMI_RH_ENDORSEMENT_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_RH_ENDORSEMENT_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                 \
              (buffer),                                                 \
              (size))
#define TPMI_RH_PROVISION_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_PROVISION_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_CLEAR_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_CLEAR_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_NV_AUTH_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_NV_AUTH_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_LOCKOUT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_LOCKOUT_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_NV_INDEX_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_NV_INDEX_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_NV_INDEX_Marshal(source, buffer, size) \
    Marshal(TPMI_RH_NV_INDEX_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_RH_NV_DEFINED_INDEX_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_NV_DEFINED_INDEX_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_NV_DEFINED_INDEX_Marshal(source, buffer, size) \
    Marshal(TPMI_RH_NV_DEFINED_INDEX_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_RH_NV_LEGACY_INDEX_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_NV_LEGACY_INDEX_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_NV_LEGACY_INDEX_Marshal(source, buffer, size) \
```

```
    Marshal(TPMI_RH_NV_LEGACY_INDEX_MARSHAL_REF, (source), (buffer), (size)
#define TPMI_RH_NV_EXP_INDEX_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_NV_DEFINED_INDEX_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_NV_EXP_INDEX_Marshal(source, buffer, size) \
    Marshal(TPMI_RH_NV_DEFINED_INDEX_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_RH_AC_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_AC_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_ACT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RH_ACT_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RH_ACT_Marshal(source, buffer, size) \
    Marshal(TPMI_RH_ACT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_HASH_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_HASH_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                           \
              (buffer),                                           \
              (size))
#define TPMI_ALG_HASH_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_HASH_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_ASYM_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_ASYM_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                           \
              (buffer),                                           \
              (size))
#define TPMI_ALG_ASYM_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_ASYM_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_SYM_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_SYM_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                          \
              (buffer),                                          \
              (size))
#define TPMI_ALG_SYM_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_SYM_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_SYM_OBJECT_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_SYM_OBJECT_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                 \
              (buffer),                                                 \
              (size))
#define TPMI_ALG_SYM_OBJECT_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_SYM_OBJECT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_SYM_MODE_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_SYM_MODE_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                               \
              (buffer),                                               \
              (size))
#define TPMI_ALG_SYM_MODE_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_SYM_MODE_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_KDF_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_KDF_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                          \
              (buffer),                                          \
              (size))
#define TPMI_ALG_KDF_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_KDF_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_SIG_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_SIG_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                 \
              (buffer),                                                 \
              (size))
#define TPMI_ALG_SIG_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_SIG_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ECC_KEY_EXCHANGE_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ECC_KEY_EXCHANGE_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                   \
              (buffer),                                                   \
              (size))
#define TPMI_ECC_KEY_EXCHANGE_Marshal(source, buffer, size) \
```

```
    Marshal(TPMI_ECC_KEY_EXCHANGE_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ST_COMMAND_TAG_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_ST_COMMAND_TAG_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_ST_COMMAND_TAG_Marshal(source, buffer, size) \
    Marshal(TPMI_ST_COMMAND_TAG_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_MAC_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_MAC_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                 \
              (buffer),                                                 \
              (size))
#define TPMI_ALG_MAC_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_MAC_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_CIPHER_MODE_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_CIPHER_MODE_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                  \
              (buffer),                                                  \
              (size))
#define TPMI_ALG_CIPHER_MODE_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_CIPHER_MODE_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_EMPTY_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_EMPTY_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_EMPTY_Marshal(source, buffer, size) \
    Marshal(TPMS_EMPTY_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ALGORITHM_DESCRIPTION_Marshal(source, buffer, size) \
    Marshal(TPMS_ALGORITHM_DESCRIPTION_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_HA_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(TPMU_HA_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_HA_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_HA_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_HA_Unmarshal(target, buffer, size, flag) \
    Unmarshal(                                        \
        TPMT_HA_MARSHAL_REF | (flag ? NULL_FLAG : 0), (target), (buffer), (size))
#define TPMT_HA_Marshal(source, buffer, size) \
    Marshal(TPMT_HA_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_DIGEST_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_DIGEST_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_DIGEST_Marshal(source, buffer, size) \
    Marshal(TPM2B_DIGEST_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_DATA_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_DATA_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_DATA_Marshal(source, buffer, size) \
    Marshal(TPM2B_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_NONCE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_NONCE_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_NONCE_Marshal(source, buffer, size) \
    Marshal(TPM2B_NONCE_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_AUTH_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_AUTH_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_AUTH_Marshal(source, buffer, size) \
    Marshal(TPM2B_AUTH_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_OPERAND_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_OPERAND_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_OPERAND_Marshal(source, buffer, size) \
    Marshal(TPM2B_OPERAND_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_EVENT_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_EVENT_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_EVENT_Marshal(source, buffer, size) \
    Marshal(TPM2B_EVENT_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_MAX_BUFFER_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_MAX_BUFFER_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_MAX_BUFFER_Marshal(source, buffer, size) \
    Marshal(TPM2B_MAX_BUFFER_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_MAX_NV_BUFFER_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_MAX_NV_BUFFER_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_MAX_NV_BUFFER_Marshal(source, buffer, size) \
    Marshal(TPM2B_MAX_NV_BUFFER_MARSHAL_REF, (source), (buffer), (size))
```

```
#define TPM2B_TIMEOUT_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_TIMEOUT_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_TIMEOUT_Marshal(source, buffer, size) \
    Marshal(TPM2B_TIMEOUT_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_IV_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_IV_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_IV_Marshal(source, buffer, size) \
    Marshal(TPM2B_IV_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_NAME_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_NAME_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_NAME_Marshal(source, buffer, size) \
    Marshal(TPM2B_NAME_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_PCR_SELECT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_PCR_SELECT_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_PCR_SELECT_Marshal(source, buffer, size) \
    Marshal(TPMS_PCR_SELECT_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_PCR_SELECTION_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_PCR_SELECTION_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_PCR_SELECTION_Marshal(source, buffer, size) \
    Marshal(TPMS_PCR_SELECTION_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_TK_CREATION_Unmarshal(target, buffer, size) \
    Unmarshal(TPMT_TK_CREATION_MARSHAL_REF, (target), (buffer), (size))
#define TPMT_TK_CREATION_Marshal(source, buffer, size) \
    Marshal(TPMT_TK_CREATION_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_TK_VERIFIED_Unmarshal(target, buffer, size) \
    Unmarshal(TPMT_TK_VERIFIED_MARSHAL_REF, (target), (buffer), (size))
#define TPMT_TK_VERIFIED_Marshal(source, buffer, size) \
    Marshal(TPMT_TK_VERIFIED_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_TK_AUTH_Unmarshal(target, buffer, size) \
    Unmarshal(TPMT_TK_AUTH_MARSHAL_REF, (target), (buffer), (size))
#define TPMT_TK_AUTH_Marshal(source, buffer, size) \
    Marshal(TPMT_TK_AUTH_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_TK_HASHCHECK_Unmarshal(target, buffer, size) \
    Unmarshal(TPMT_TK_HASHCHECK_MARSHAL_REF, (target), (buffer), (size))
#define TPMT_TK_HASHCHECK_Marshal(source, buffer, size) \
    Marshal(TPMT_TK_HASHCHECK_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ALG_PROPERTY_Marshal(source, buffer, size) \
    Marshal(TPMS_ALG_PROPERTY_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_TAGGED_PROPERTY_Marshal(source, buffer, size) \
    Marshal(TPMS_TAGGED_PROPERTY_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_TAGGED_PCR_SELECT_Marshal(source, buffer, size) \
    Marshal(TPMS_TAGGED_PCR_SELECT_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_TAGGED_POLICY_Marshal(source, buffer, size) \
    Marshal(TPMS_TAGGED_POLICY_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ACT_DATA_Marshal(source, buffer, size) \
    Marshal(TPMS_ACT_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPML_CC_Unmarshal(target, buffer, size) \
    Unmarshal(TPML_CC_MARSHAL_REF, (target), (buffer), (size))
#define TPML_CC_Marshal(source, buffer, size) \
    Marshal(TPML_CC_MARSHAL_REF, (source), (buffer), (size))
#define TPML_CCA_Marshal(source, buffer, size) \
    Marshal(TPML_CCA_MARSHAL_REF, (source), (buffer), (size))
#define TPML_ALG_Unmarshal(target, buffer, size) \
    Unmarshal(TPML_ALG_MARSHAL_REF, (target), (buffer), (size))
#define TPML_ALG_Marshal(source, buffer, size) \
    Marshal(TPML_ALG_MARSHAL_REF, (source), (buffer), (size))
#define TPML_HANDLE_Marshal(source, buffer, size) \
    Marshal(TPML_HANDLE_MARSHAL_REF, (source), (buffer), (size))
#define TPML_DIGEST_Unmarshal(target, buffer, size) \
    Unmarshal(TPML_DIGEST_MARSHAL_REF, (target), (buffer), (size))
#define TPML_DIGEST_Marshal(source, buffer, size) \
    Marshal(TPML_DIGEST_MARSHAL_REF, (source), (buffer), (size))
#define TPML_DIGEST_VALUES_Unmarshal(target, buffer, size) \
    Unmarshal(TPML_DIGEST_VALUES_MARSHAL_REF, (target), (buffer), (size))
#define TPML_DIGEST_VALUES_Marshal(source, buffer, size) \
    Marshal(TPML_DIGEST_VALUES_MARSHAL_REF, (source), (buffer), (size))
```

```
#define TPML_PCR_SELECTION_Unmarshal(target, buffer, size) \
    Unmarshal(TPML_PCR_SELECTION_MARSHAL_REF, (target), (buffer), (size))
#define TPML_PCR_SELECTION_Marshal(source, buffer, size) \
    Marshal(TPML_PCR_SELECTION_MARSHAL_REF, (source), (buffer), (size))
#define TPML_ALG_PROPERTY_Marshal(source, buffer, size) \
    Marshal(TPML_ALG_PROPERTY_MARSHAL_REF, (source), (buffer), (size))
#define TPML_TAGGED_TPM_PROPERTY_Marshal(source, buffer, size) \
    Marshal(TPML_TAGGED_TPM_PROPERTY_MARSHAL_REF, (source), (buffer), (size))
#define TPML_TAGGED_PCR_PROPERTY_Marshal(source, buffer, size) \
    Marshal(TPML_TAGGED_PCR_PROPERTY_MARSHAL_REF, (source), (buffer), (size))
#define TPML_ECC_CURVE_Marshal(source, buffer, size) \
    Marshal(TPML_ECC_CURVE_MARSHAL_REF, (source), (buffer), (size))
#define TPML_TAGGED_POLICY_Marshal(source, buffer, size) \
    Marshal(TPML_TAGGED_POLICY_MARSHAL_REF, (source), (buffer), (size))
#define TPML_ACT_DATA_Marshal(source, buffer, size) \
    Marshal(TPML_ACT_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_CAPABILITIES_Marshal(source, buffer, size, selector) \
    MarshalUnion(                                                 \
        TPMU_CAPABILITIES_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMS_CAPABILITY_DATA_Marshal(source, buffer, size) \
    Marshal(TPMS_CAPABILITY_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_SET_CAPABILITIES_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                                     \
        TPMU_SET_CAPABILITIES_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_SET_CAPABILITIES_Marshal(source, buffer, size, selector) \
    MarshalUnion(                                                     \
        TPMU_SET_CAPABILITIES_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMS_SET_CAPABILITY_DATA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SET_CAPABILITY_DATA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SET_CAPABILITY_DATA_Marshal(source, buffer, size) \
    Marshal(TPMS_SET_CAPABILITY_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_SET_CAPABILITY_DATA_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_SET_CAPABILITY_DATA_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_SET_CAPABILITY_DATA_Marshal(source, buffer, size) \
    Marshal(TPM2B_SET_CAPABILITY_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_CLOCK_INFO_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_CLOCK_INFO_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_CLOCK_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_CLOCK_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_TIME_INFO_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_TIME_INFO_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_TIME_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_TIME_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_TIME_ATTEST_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_TIME_ATTEST_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_CERTIFY_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_CERTIFY_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_QUOTE_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_QUOTE_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_COMMAND_AUDIT_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_COMMAND_AUDIT_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SESSION_AUDIT_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_SESSION_AUDIT_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_CREATION_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_CREATION_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_NV_CERTIFY_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_NV_CERTIFY_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_NV_DIGEST_CERTIFY_INFO_Marshal(source, buffer, size) \
    Marshal(TPMS_NV_DIGEST_CERTIFY_INFO_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ST_ATTEST_Marshal(source, buffer, size) \
    Marshal(TPMI_ST_ATTEST_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_ATTEST_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_ATTEST_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMS_ATTEST_Marshal(source, buffer, size) \
    Marshal(TPMS_ATTEST_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_ATTEST_Marshal(source, buffer, size) \
```

```
    Marshal(TPM2B_ATTEST_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_AUTH_COMMAND_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_AUTH_COMMAND_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_AUTH_RESPONSE_Marshal(source, buffer, size) \
    Marshal(TPMS_AUTH_RESPONSE_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_TDES_KEY_BITS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_TDES_KEY_BITS_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_TDES_KEY_BITS_Marshal(source, buffer, size) \
    Marshal(TPMI_TDES_KEY_BITS_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_AES_KEY_BITS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_AES_KEY_BITS_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_AES_KEY_BITS_Marshal(source, buffer, size) \
    Marshal(TPMI_AES_KEY_BITS_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_SM4_KEY_BITS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_SM4_KEY_BITS_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_SM4_KEY_BITS_Marshal(source, buffer, size) \
    Marshal(TPMI_SM4_KEY_BITS_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_CAMELLIA_KEY_BITS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_CAMELLIA_KEY_BITS_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_CAMELLIA_KEY_BITS_Marshal(source, buffer, size) \
    Marshal(TPMI_CAMELLIA_KEY_BITS_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_SYM_KEY_BITS_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                                 \
        TPMU_SYM_KEY_BITS_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_SYM_KEY_BITS_Marshal(source, buffer, size, selector) \
    MarshalUnion(                                                 \
        TPMU_SYM_KEY_BITS_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMU_SYM_MODE_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(TPMU_SYM_MODE_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_SYM_MODE_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_SYM_MODE_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_SYM_DEF_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_SYM_DEF_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                          \
              (buffer),                                          \
              (size))
#define TPMT_SYM_DEF_Marshal(source, buffer, size) \
    Marshal(TPMT_SYM_DEF_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_SYM_DEF_OBJECT_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_SYM_DEF_OBJECT_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                 \
              (buffer),                                                 \
              (size))
#define TPMT_SYM_DEF_OBJECT_Marshal(source, buffer, size) \
    Marshal(TPMT_SYM_DEF_OBJECT_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_SYM_KEY_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_SYM_KEY_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_SYM_KEY_Marshal(source, buffer, size) \
    Marshal(TPM2B_SYM_KEY_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SYMCIPHER_PARMS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SYMCIPHER_PARMS_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SYMCIPHER_PARMS_Marshal(source, buffer, size) \
    Marshal(TPMS_SYMCIPHER_PARMS_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_LABEL_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_LABEL_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_LABEL_Marshal(source, buffer, size) \
    Marshal(TPM2B_LABEL_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_DERIVE_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_DERIVE_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_DERIVE_Marshal(source, buffer, size) \
    Marshal(TPMS_DERIVE_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_DERIVE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_DERIVE_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_DERIVE_Marshal(source, buffer, size) \
    Marshal(TPM2B_DERIVE_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_SENSITIVE_DATA_Unmarshal(target, buffer, size) \
```

```
    Unmarshal(TPM2B_SENSITIVE_DATA_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_SENSITIVE_DATA_Marshal(source, buffer, size) \
    Marshal(TPM2B_SENSITIVE_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SENSITIVE_CREATE_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SENSITIVE_CREATE_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_SENSITIVE_CREATE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_SENSITIVE_CREATE_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SCHEME_HASH_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SCHEME_HASH_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SCHEME_HASH_Marshal(source, buffer, size) \
    Marshal(TPMS_SCHEME_HASH_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SCHEME_ECDAA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SCHEME_ECDAA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SCHEME_ECDAA_Marshal(source, buffer, size) \
    Marshal(TPMS_SCHEME_ECDAA_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_KEYEDHASH_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_KEYEDHASH_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                       \
              (buffer),                                                       \
              (size))
#define TPMI_ALG_KEYEDHASH_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_KEYEDHASH_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SCHEME_HMAC_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SCHEME_HMAC_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SCHEME_HMAC_Marshal(source, buffer, size) \
    Marshal(TPMS_SCHEME_HMAC_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SCHEME_XOR_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SCHEME_XOR_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SCHEME_XOR_Marshal(source, buffer, size) \
    Marshal(TPMS_SCHEME_XOR_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_SCHEME_KEYEDHASH_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                                     \
        TPMU_SCHEME_KEYEDHASH_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_SCHEME_KEYEDHASH_Marshal(source, buffer, size, selector) \
    MarshalUnion(                                                     \
        TPMU_SCHEME_KEYEDHASH_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_KEYEDHASH_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_KEYEDHASH_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                   \
              (buffer),                                                   \
              (size))
#define TPMT_KEYEDHASH_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMT_KEYEDHASH_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIG_SCHEME_RSASSA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIG_SCHEME_RSASSA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIG_SCHEME_RSASSA_Marshal(source, buffer, size) \
    Marshal(TPMS_SIG_SCHEME_RSASSA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIG_SCHEME_RSAPSS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIG_SCHEME_RSAPSS_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIG_SCHEME_RSAPSS_Marshal(source, buffer, size) \
    Marshal(TPMS_SIG_SCHEME_RSAPSS_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIG_SCHEME_ECDSA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIG_SCHEME_ECDSA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIG_SCHEME_ECDSA_Marshal(source, buffer, size) \
    Marshal(TPMS_SIG_SCHEME_ECDSA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIG_SCHEME_SM2_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIG_SCHEME_SM2_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIG_SCHEME_SM2_Marshal(source, buffer, size) \
    Marshal(TPMS_SIG_SCHEME_SM2_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIG_SCHEME_ECSCHNORR_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIG_SCHEME_ECSCHNORR_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIG_SCHEME_ECSCHNORR_Marshal(source, buffer, size) \
    Marshal(TPMS_SIG_SCHEME_ECSCHNORR_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIG_SCHEME_ECDAA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIG_SCHEME_ECDAA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIG_SCHEME_ECDAA_Marshal(source, buffer, size) \
```

```
    Marshal(TPMS_SIG_SCHEME_ECDAA_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_SIG_SCHEME_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                               \
        TPMU_SIG_SCHEME_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_SIG_SCHEME_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_SIG_SCHEME_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_SIG_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_SIG_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                             \
              (buffer),                                             \
              (size))
#define TPMT_SIG_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMT_SIG_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ENC_SCHEME_OAEP_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_ENC_SCHEME_OAEP_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_ENC_SCHEME_OAEP_Marshal(source, buffer, size) \
    Marshal(TPMS_ENC_SCHEME_OAEP_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ENC_SCHEME_RSAES_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_ENC_SCHEME_RSAES_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_ENC_SCHEME_RSAES_Marshal(source, buffer, size) \
    Marshal(TPMS_ENC_SCHEME_RSAES_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_KEY_SCHEME_ECDH_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_KEY_SCHEME_ECDH_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_KEY_SCHEME_ECDH_Marshal(source, buffer, size) \
    Marshal(TPMS_KEY_SCHEME_ECDH_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_KEY_SCHEME_ECMQV_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_KEY_SCHEME_ECMQV_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_KEY_SCHEME_ECMQV_Marshal(source, buffer, size) \
    Marshal(TPMS_KEY_SCHEME_ECMQV_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_KDF_SCHEME_MGF1_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_KDF_SCHEME_MGF1_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_KDF_SCHEME_MGF1_Marshal(source, buffer, size) \
    Marshal(TPMS_KDF_SCHEME_MGF1_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_KDF_SCHEME_KDF1_SP800_56A_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_KDF_SCHEME_KDF1_SP800_56A_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_KDF_SCHEME_KDF1_SP800_56A_Marshal(source, buffer, size) \
    Marshal(TPMS_KDF_SCHEME_KDF1_SP800_56A_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_KDF_SCHEME_KDF2_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_KDF_SCHEME_KDF2_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_KDF_SCHEME_KDF2_Marshal(source, buffer, size) \
    Marshal(TPMS_KDF_SCHEME_KDF2_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_KDF_SCHEME_KDF1_SP800_108_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_KDF_SCHEME_KDF1_SP800_108_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_KDF_SCHEME_KDF1_SP800_108_Marshal(source, buffer, size) \
    Marshal(TPMS_KDF_SCHEME_KDF1_SP800_108_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_KDF_SCHEME_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                               \
        TPMU_KDF_SCHEME_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_KDF_SCHEME_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_KDF_SCHEME_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_KDF_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_KDF_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                             \
              (buffer),                                             \
              (size))
#define TPMT_KDF_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMT_KDF_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_ASYM_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_ASYM_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                  \
              (buffer),                                                  \
              (size))
#define TPMI_ALG_ASYM_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_ASYM_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_ASYM_SCHEME_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                                \
```

```
        TPMU_ASYM_SCHEME_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_ASYM_SCHEME_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_ASYM_SCHEME_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMI_ALG_RSA_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_RSA_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                 \
              (buffer),                                                 \
              (size))
#define TPMI_ALG_RSA_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_RSA_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_RSA_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_RSA_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                             \
              (buffer),                                             \
              (size))
#define TPMT_RSA_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMT_RSA_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_RSA_DECRYPT_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_RSA_DECRYPT_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                  \
              (buffer),                                                  \
              (size))
#define TPMI_ALG_RSA_DECRYPT_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_RSA_DECRYPT_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_RSA_DECRYPT_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_RSA_DECRYPT_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                              \
              (buffer),                                              \
              (size))
#define TPMT_RSA_DECRYPT_Marshal(source, buffer, size) \
    Marshal(TPMT_RSA_DECRYPT_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_PUBLIC_KEY_RSA_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_PUBLIC_KEY_RSA_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_PUBLIC_KEY_RSA_Marshal(source, buffer, size) \
    Marshal(TPM2B_PUBLIC_KEY_RSA_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_RSA_KEY_BITS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMI_RSA_KEY_BITS_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_RSA_KEY_BITS_Marshal(source, buffer, size) \
    Marshal(TPMI_RSA_KEY_BITS_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_PRIVATE_KEY_RSA_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_PRIVATE_KEY_RSA_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_PRIVATE_KEY_RSA_Marshal(source, buffer, size) \
    Marshal(TPM2B_PRIVATE_KEY_RSA_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_ECC_PARAMETER_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_ECC_PARAMETER_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_ECC_PARAMETER_Marshal(source, buffer, size) \
    Marshal(TPM2B_ECC_PARAMETER_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ECC_POINT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_ECC_POINT_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_ECC_POINT_Marshal(source, buffer, size) \
    Marshal(TPMS_ECC_POINT_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_ECC_POINT_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_ECC_POINT_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_ECC_POINT_Marshal(source, buffer, size) \
    Marshal(TPM2B_ECC_POINT_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_ECC_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ALG_ECC_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                                 \
              (buffer),                                                 \
              (size))
#define TPMI_ALG_ECC_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_ECC_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ECC_CURVE_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMI_ECC_CURVE_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                            \
              (buffer),                                            \
```

```
            (size))
#define TPMI_ECC_CURVE_Marshal(source, buffer, size) \
    Marshal(TPMI_ECC_CURVE_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_ECC_SCHEME_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_ECC_SCHEME_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
            (target),                                               \
            (buffer),                                               \
            (size))
#define TPMT_ECC_SCHEME_Marshal(source, buffer, size) \
    Marshal(TPMT_ECC_SCHEME_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ALGORITHM_DETAIL_ECC_Marshal(source, buffer, size) \
    Marshal(TPMS_ALGORITHM_DETAIL_ECC_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_RSA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_RSA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_RSA_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_RSA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_RSASSA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_RSASSA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_RSASSA_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_RSASSA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_RSAPSS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_RSAPSS_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_RSAPSS_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_RSAPSS_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_ECC_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_ECC_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_ECC_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_ECC_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_ECDAA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_ECDAA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_ECDAA_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_ECDAA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_ECDSA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_ECDSA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_ECDSA_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_ECDSA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_SM2_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_SM2_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_SM2_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_SM2_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_SIGNATURE_ECSCHNORR_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_SIGNATURE_ECSCHNORR_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_SIGNATURE_ECSCHNORR_Marshal(source, buffer, size) \
    Marshal(TPMS_SIGNATURE_ECSCHNORR_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_SIGNATURE_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(TPMU_SIGNATURE_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_SIGNATURE_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_SIGNATURE_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_SIGNATURE_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_SIGNATURE_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
            (target),                                              \
            (buffer),                                              \
            (size))
#define TPMT_SIGNATURE_Marshal(source, buffer, size) \
    Marshal(TPMT_SIGNATURE_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_ENCRYPTED_SECRET_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                                     \
        TPMU_ENCRYPTED_SECRET_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_ENCRYPTED_SECRET_Marshal(source, buffer, size, selector) \
    MarshalUnion(                                                     \
        TPMU_ENCRYPTED_SECRET_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPM2B_ENCRYPTED_SECRET_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_ENCRYPTED_SECRET_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_ENCRYPTED_SECRET_Marshal(source, buffer, size) \
    Marshal(TPM2B_ENCRYPTED_SECRET_MARSHAL_REF, (source), (buffer), (size))
#define TPMI_ALG_PUBLIC_Unmarshal(target, buffer, size) \
```

```
    Unmarshal(TPMI_ALG_PUBLIC_MARSHAL_REF, (target), (buffer), (size))
#define TPMI_ALG_PUBLIC_Marshal(source, buffer, size) \
    Marshal(TPMI_ALG_PUBLIC_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_PUBLIC_ID_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(TPMU_PUBLIC_ID_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_PUBLIC_ID_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_PUBLIC_ID_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMS_KEYEDHASH_PARMS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_KEYEDHASH_PARMS_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_KEYEDHASH_PARMS_Marshal(source, buffer, size) \
    Marshal(TPMS_KEYEDHASH_PARMS_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_RSA_PARMS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_RSA_PARMS_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_RSA_PARMS_Marshal(source, buffer, size) \
    Marshal(TPMS_RSA_PARMS_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_ECC_PARMS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_ECC_PARMS_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_ECC_PARMS_Marshal(source, buffer, size) \
    Marshal(TPMS_ECC_PARMS_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_PUBLIC_PARMS_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                                 \
        TPMU_PUBLIC_PARMS_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_PUBLIC_PARMS_Marshal(source, buffer, size, selector) \
    MarshalUnion(                                                 \
        TPMU_PUBLIC_PARMS_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_PUBLIC_PARMS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMT_PUBLIC_PARMS_MARSHAL_REF, (target), (buffer), (size))
#define TPMT_PUBLIC_PARMS_Marshal(source, buffer, size) \
    Marshal(TPMT_PUBLIC_PARMS_MARSHAL_REF, (source), (buffer), (size))
#define TPMT_PUBLIC_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPMT_PUBLIC_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                         \
              (buffer),                                         \
              (size))
#define TPMT_PUBLIC_Marshal(source, buffer, size) \
    Marshal(TPMT_PUBLIC_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_PUBLIC_Unmarshal(target, buffer, size, flag)       \
    Unmarshal(TPM2B_PUBLIC_MARSHAL_REF | (flag ? NULL_FLAG : 0), \
              (target),                                          \
              (buffer),                                          \
              (size))
#define TPM2B_PUBLIC_Marshal(source, buffer, size) \
    Marshal(TPM2B_PUBLIC_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_TEMPLATE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_TEMPLATE_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_TEMPLATE_Marshal(source, buffer, size) \
    Marshal(TPM2B_TEMPLATE_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_PRIVATE_VENDOR_SPECIFIC_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_PRIVATE_VENDOR_SPECIFIC_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_PRIVATE_VENDOR_SPECIFIC_Marshal(source, buffer, size) \
    Marshal(TPM2B_PRIVATE_VENDOR_SPECIFIC_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_SENSITIVE_COMPOSITE_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(TPMU_SENSITIVE_COMPOSITE_MARSHAL_REF,                   \
                   (target),                                               \
                   (buffer),                                               \
                   (size),                                                 \
                   (selector))
#define TPMU_SENSITIVE_COMPOSITE_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_SENSITIVE_COMPOSITE_MARSHAL_REF,                   \
                 (source),                                               \
                 (buffer),                                               \
                 (size),                                                 \
                 (selector))
#define TPMT_SENSITIVE_Unmarshal(target, buffer, size) \
    Unmarshal(TPMT_SENSITIVE_MARSHAL_REF, (target), (buffer), (size))
#define TPMT_SENSITIVE_Marshal(source, buffer, size) \
```

```
    Marshal(TPMT_SENSITIVE_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_SENSITIVE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_SENSITIVE_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_SENSITIVE_Marshal(source, buffer, size) \
    Marshal(TPM2B_SENSITIVE_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_PRIVATE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_PRIVATE_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_PRIVATE_Marshal(source, buffer, size) \
    Marshal(TPM2B_PRIVATE_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_ID_OBJECT_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_ID_OBJECT_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_ID_OBJECT_Marshal(source, buffer, size) \
    Marshal(TPM2B_ID_OBJECT_MARSHAL_REF, (source), (buffer), (size))
#define TPM_NV_INDEX_Marshal(source, buffer, size) \
    Marshal(TPM_NV_INDEX_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_NV_PIN_COUNTER_PARAMETERS_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_NV_PIN_COUNTER_PARAMETERS_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_NV_PIN_COUNTER_PARAMETERS_Marshal(source, buffer, size) \
    Marshal(TPMS_NV_PIN_COUNTER_PARAMETERS_MARSHAL_REF, (source), (buffer), (size))
#define TPMA_NV_Unmarshal(target, buffer, size) \
    Unmarshal(TPMA_NV_MARSHAL_REF, (target), (buffer), (size))
#define TPMA_NV_Marshal(source, buffer, size) \
    Marshal(TPMA_NV_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_NV_PUBLIC_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_NV_PUBLIC_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_NV_PUBLIC_Marshal(source, buffer, size) \
    Marshal(TPMS_NV_PUBLIC_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_NV_PUBLIC_EXP_ATTR_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_NV_PUBLIC_EXP_ATTR_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_NV_PUBLIC_EXP_ATTR_Marshal(source, buffer, size) \
    Marshal(TPMS_NV_PUBLIC_EXP_ATTR_MARSHAL_REF, (source), (buffer), (size))
#define TPMU_NV_PUBLIC_2_Unmarshal(target, buffer, size, selector) \
    UnmarshalUnion(                                                \
        TPMU_NV_PUBLIC_2_MARSHAL_REF, (target), (buffer), (size), (selector))
#define TPMU_NV_PUBLIC_2_Marshal(source, buffer, size, selector) \
    MarshalUnion(TPMU_NV_PUBLIC_2_MARSHAL_REF, (source), (buffer), (size), (selector))
#define TPMT_NV_PUBLIC_2_Unmarshal(target, buffer, size) \
    Unmarshal(TPMT_NV_PUBLIC_2_MARSHAL_REF, (target), (buffer), (size))
#define TPMT_NV_PUBLIC_2_Marshal(source, buffer, size) \
    Marshal(TPMT_NV_PUBLIC_2_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_NV_PUBLIC_2_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_NV_PUBLIC_2_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_NV_PUBLIC_2_Marshal(source, buffer, size) \
    Marshal(TPM2B_NV_PUBLIC_2_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_NV_PUBLIC_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_NV_PUBLIC_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_NV_PUBLIC_Marshal(source, buffer, size) \
    Marshal(TPM2B_NV_PUBLIC_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_CONTEXT_SENSITIVE_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_CONTEXT_SENSITIVE_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_CONTEXT_SENSITIVE_Marshal(source, buffer, size) \
    Marshal(TPM2B_CONTEXT_SENSITIVE_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_CONTEXT_DATA_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_CONTEXT_DATA_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_CONTEXT_DATA_Marshal(source, buffer, size) \
    Marshal(TPMS_CONTEXT_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPM2B_CONTEXT_DATA_Unmarshal(target, buffer, size) \
    Unmarshal(TPM2B_CONTEXT_DATA_MARSHAL_REF, (target), (buffer), (size))
#define TPM2B_CONTEXT_DATA_Marshal(source, buffer, size) \
    Marshal(TPM2B_CONTEXT_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_CONTEXT_Unmarshal(target, buffer, size) \
    Unmarshal(TPMS_CONTEXT_MARSHAL_REF, (target), (buffer), (size))
#define TPMS_CONTEXT_Marshal(source, buffer, size) \
    Marshal(TPMS_CONTEXT_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_CREATION_DATA_Marshal(source, buffer, size) \
    Marshal(TPMS_CREATION_DATA_MARSHAL_REF, (source), (buffer), (size))
```

```
#define TPM2B_CREATION_DATA_Marshal(source, buffer, size) \
    Marshal(TPM2B_CREATION_DATA_MARSHAL_REF, (source), (buffer), (size))
#define TPM_AT_Unmarshal(target, buffer, size) \
    Unmarshal(TPM_AT_MARSHAL_REF, (target), (buffer), (size))
#define TPM_AT_Marshal(source, buffer, size) \
    Marshal(TPM_AT_MARSHAL_REF, (source), (buffer), (size))
#define TPM_AE_Marshal(source, buffer, size) \
    Marshal(TPM_AE_MARSHAL_REF, (source), (buffer), (size))
#define TPMS_AC_OUTPUT_Marshal(source, buffer, size) \
    Marshal(TPMS_AC_OUTPUT_MARSHAL_REF, (source), (buffer), (size))
#define TPML_AC_CAPABILITIES_Marshal(source, buffer, size) \
    Marshal(TPML_AC_CAPABILITIES_MARSHAL_REF, (source), (buffer), (size))

#endif  // _TABLE_MARSHAL_DEFINES_H_
```

6.39 /tpm/include/private/TableMarshalTypes.h

```
#ifndef _TABLE_MARSHAL_TYPES_H_
#define _TABLE_MARSHAL_TYPES_H_

typedef UINT16 marshalIndex_t;

//*** Structure Entries
// A structure contains a list of elements to unmarshal. Each of the entries is a
// UINT16. The structure descriptor is:

// The 'values' array contains indicators for the things to marshal. The 'elements'
// parameter indicates how many different entities are unmarshaled. This number
// nominally corresponds to the number of rows in the Part 2 table that describes
// the structure (the number of rows minus the title row and any error code rows).

// A schematic of a simple structure entry is shown here but the values are not
// actually in a structure. As shown, the third value is the offset in the structure
// where the value is placed when unmarshaled, or fetched from when marshaling. This
// is sufficient when the element type indicated by 'index' is always a simple type
// and never a union or array.This is just shown for illustrative purposes.
typedef struct simpleStructureEntry_t
{
    UINT16 qualifiers;     // indicates the type of entry (array, union
                           // etc.)
    marshalIndex_t index;  // the index into the appropriate array of
                           //  the descriptor of this type
    UINT16 offset;         // where this comes from or is placed
} simpleStructureEntry_t;

typedef const struct UintMarshal_mst
{
    UINT8 marshalType;  // UINT_MTYPE
    UINT8 modifiers;    // size and signed indicator.
} UintMarshal_mst;

typedef struct UnionMarshal_mst
{
    UINT8  countOfselectors;
    UINT8  modifiers;  // NULL_SELECTOR
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[1];
    //    UINT16          marshalingTypes[1]; // This is not part of the prototypical
    //    entry. It is here to show where the
    //    marshaling types will be in a union
} UnionMarshal_mst;

typedef struct NullUnionMarshal_mst
{
    UINT8 count;
```

```
} NullUnionMarshal_mst;

typedef struct MarshalHeader_mst
{
    UINT8 marshalType;  // VALUES_MTYPE
    UINT8 modifiers;
    UINT8 errorCode;
} MarshalHeader_mst;

typedef const struct ArrayMarshal_mst  // used in a structure
{
    marshalIndex_t type;
    UINT16         stride;
} ArrayMarshal_mst;

typedef const struct StructMarshal_mst
{
    UINT8  marshalType;  // STRUCTURE_MTYPE
    UINT8  elements;
    UINT16 values[1];  // three times elements
} StructMarshal_mst;

typedef const struct ValuesMarshal_mst
{
    UINT8  marshalType;  // VALUES_MTYPE
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[1];
} ValuesMarshal_mst;

typedef const struct TableMarshal_mst
{
    UINT8  marshalType;  // TABLE_MTYPE
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  singles;
    UINT32 values[1];
} TableMarshal_mst;

typedef const struct MinMaxMarshal_mst
{
    UINT8  marshalType;  // MIN_MAX_MTYPE
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} MinMaxMarshal_mst;

typedef const struct Tpm2bMarshal_mst
{
    UINT8  unmarshalType;  // TPM2B_MTYPE
    UINT16 sizeIndex;      // reference to type for this size value
} Tpm2bMarshal_mst;

typedef const struct Tpm2bsMarshal_mst
{
    UINT8  unmarshalType;  // TPM2BS_MTYPE
    UINT8  modifiers;      // size= and offset (2 - 7)
    UINT16 sizeIndex;      // index of the size value;
    UINT16 dataIndex;      // the structure
} Tpm2bsMarshal_mst;

typedef const struct ListMarshal_mst
{
    UINT8  unmarshalType;  // LIST_MTYPE (for TPML)
```

```
    UINT8  modifiers;      // size offset 2-7
    UINT16 sizeIndex;      // reference to the minmax structure that
                           //      unmarshals the size parameter
    UINT16 arrayRef;       // reference to an array definition (type
                           //  and stride)
} ListMarshal_mst;

typedef const struct AttributesMarshal_mst
{
    UINT8  unmarashalType;  // ATTRIBUTE_MTYPE
    UINT8  modifiers;       // size (ONE_BYTES, TWO_BYTES, or FOUR_BYTES
    UINT32 attributeMask;   // the values that must be zero.
} AttributesMarshal_mst;

typedef const struct Attributes64Marshal_mst
{
    UINT8  unmarashalType;  // ATTRIBUTE_MTYPE
    UINT8  modifiers;       // size (ONE_BYTES, TWO_BYTES, or FOUR_BYTES
    UINT64 attributeMask;   // the values that must be zero.
} Attributes64Marshal_mst;

typedef const struct CompositeMarshal_mst
{
    UINT8          unmashalType;  // COMPOSITE_MTYPE
    UINT8          modifiers;     // number of entries and size
    marshalIndex_t types[1];      // array of unmarshaling types
} CompositeMarshal_mst;

typedef const struct TPM_ECC_CURVE_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPM_ECC_CURVE_mst;

typedef const struct TPM_CLOCK_ADJUST_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPM_CLOCK_ADJUST_mst;

typedef const struct TPM_EO_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPM_EO_mst;

typedef const struct TPM_SU_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} TPM_SU_mst;

typedef const struct TPM_SE_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
```

```
    UINT8  entries;
    UINT32 values[3];
} TPM_SE_mst;

typedef const struct TPM_CAP_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[3];
} TPM_CAP_mst;

typedef const struct TPMI_YES_NO_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} TPMI_YES_NO_mst;

typedef const struct TPMI_DH_OBJECT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[5];
} TPMI_DH_OBJECT_mst;

typedef const struct TPMI_DH_PARENT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[20];
} TPMI_DH_PARENT_mst;

typedef const struct TPMI_DH_PERSISTENT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPMI_DH_PERSISTENT_mst;

typedef const struct TPMI_DH_ENTITY_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[15];
} TPMI_DH_ENTITY_mst;

typedef const struct TPMI_DH_PCR_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
```

```
    UINT32 values[3];
} TPMI_DH_PCR_mst;

typedef const struct TPMI_SH_AUTH_SESSION_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[5];
} TPMI_SH_AUTH_SESSION_mst;

typedef const struct TPMI_SH_HMAC_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPMI_SH_HMAC_mst;

typedef const struct TPMI_SH_POLICY_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPMI_SH_POLICY_mst;

typedef const struct TPMI_DH_CONTEXT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[6];
} TPMI_DH_CONTEXT_mst;

typedef const struct TPMI_DH_SAVED_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[7];
} TPMI_DH_SAVED_mst;

typedef const struct TPMI_RH_HIERARCHY_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[16];
} TPMI_RH_HIERARCHY_mst;

typedef const struct TPMI_RH_ENABLES_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[5];
```

```
} TPMI_RH_ENABLES_mst;

typedef const struct TPMI_RH_HIERARCHY_AUTH_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[4];
} TPMI_RH_HIERARCHY_AUTH_mst;

typedef const struct TPMI_RH_HIERARCHY_POLICY_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[6];
} TPMI_RH_HIERARCHY_POLICY_mst;

typedef const struct TPMI_RH_BASE_HIERARCHY_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[3];
} TPMI_RH_BASE_HIERARCHY_mst;

typedef const struct TPMI_RH_PLATFORM_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[1];
} TPMI_RH_PLATFORM_mst;

typedef const struct TPMI_RH_OWNER_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} TPMI_RH_OWNER_mst;

typedef const struct TPMI_RH_ENDORSEMENT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} TPMI_RH_ENDORSEMENT_mst;

typedef const struct TPMI_RH_PROVISION_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} TPMI_RH_PROVISION_mst;
```

```
typedef const struct TPMI_RH_CLEAR_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} TPMI_RH_CLEAR_mst;

typedef const struct TPMI_RH_NV_AUTH_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[4];
} TPMI_RH_NV_AUTH_mst;

typedef const struct TPMI_RH_LOCKOUT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[1];
} TPMI_RH_LOCKOUT_mst;

typedef const struct TPMI_RH_NV_INDEX_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[6];
} TPMI_RH_NV_INDEX_mst;

typedef const struct TPMI_RH_NV_DEFINED_INDEX_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[4];
} TPMI_RH_NV_DEFINED_INDEX_mst;

typedef const struct TPMI_RH_LEGACY_NV_INDEX_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPMI_RH_LEGACY_NV_INDEX_mst;

typedef const struct TPMI_RH_NV_EXP_INDEX_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPMI_RH_NV_EXP_INDEX_mst;

typedef const struct TPMI_RH_AC_mst
{
```

```
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPMI_RH_AC_mst;

typedef const struct TPMI_RH_ACT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} TPMI_RH_ACT_mst;

typedef const struct TPMI_ALG_HASH_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[5];
} TPMI_ALG_HASH_mst;

typedef const struct TPMI_ALG_ASYM_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[5];
} TPMI_ALG_ASYM_mst;

typedef const struct TPMI_ALG_SYM_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[5];
} TPMI_ALG_SYM_mst;

typedef const struct TPMI_ALG_SYM_OBJECT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[5];
} TPMI_ALG_SYM_OBJECT_mst;

typedef const struct TPMI_ALG_SYM_MODE_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_SYM_MODE_mst;

typedef const struct TPMI_ALG_KDF_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_KDF_mst;

typedef const struct TPMI_ALG_SIG_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
```

```
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_SIG_SCHEME_mst;

typedef const struct TPMI_ECC_KEY_EXCHANGE_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ECC_KEY_EXCHANGE_mst;

typedef const struct TPMI_ST_COMMAND_TAG_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} TPMI_ST_COMMAND_TAG_mst;

typedef const struct TPMI_ALG_MAC_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[5];
} TPMI_ALG_MAC_SCHEME_mst;

typedef const struct TPMI_ALG_CIPHER_MODE_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_CIPHER_MODE_mst;

typedef const struct TPMS_EMPTY_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[3];
} TPMS_EMPTY_mst;

typedef const struct TPMS_ALGORITHM_DESCRIPTION_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_ALGORITHM_DESCRIPTION_mst;

typedef struct TPMU_HA_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[9];
    UINT16 marshalingTypes[9];
} TPMU_HA_mst;

typedef const struct TPMT_HA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_HA_mst;
```

```
typedef const struct TPMS_PCR_SELECT_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_PCR_SELECT_mst;

typedef const struct TPMS_PCR_SELECTION_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMS_PCR_SELECTION_mst;

typedef const struct TPMT_TK_CREATION_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMT_TK_CREATION_mst;

typedef const struct TPMT_TK_VERIFIED_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMT_TK_VERIFIED_mst;

typedef const struct TPMT_TK_AUTH_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMT_TK_AUTH_mst;

typedef const struct TPMT_TK_HASHCHECK_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMT_TK_HASHCHECK_mst;

typedef const struct TPMS_ALG_PROPERTY_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_ALG_PROPERTY_mst;

typedef const struct TPMS_TAGGED_PROPERTY_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_TAGGED_PROPERTY_mst;

typedef const struct TPMS_TAGGED_PCR_SELECT_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMS_TAGGED_PCR_SELECT_mst;

typedef const struct TPMS_TAGGED_POLICY_mst
{
```

```
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_TAGGED_POLICY_mst;

typedef const struct TPMS_ACT_DATA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMS_ACT_DATA_mst;

typedef struct TPMU_CAPABILITIES_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[11];
    UINT16 marshalingTypes[11];
} TPMU_CAPABILITIES_mst;

typedef const struct TPMS_CAPABILITY_DATA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_CAPABILITY_DATA_mst;

typedef const struct TPMU_SET_CAPABILITIES_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[0];
    UINT16 marshalingTypes[0];
} TPMU_SET_CAPABILITIES_mst;

typedef const struct TPMS_SET_CAPABILITY_DATA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_SET_CAPABILITY_DATA_mst;

typedef const struct TPM2B_SET_CAPABILITY_DATA_mst
{
    UINT8  marshalType;
    UINT16 sizeIndex;
} TPM2B_SET_CAPABILITY_DATA_mst;

typedef const struct TPMS_CLOCK_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[12];
} TPMS_CLOCK_INFO_mst;

typedef const struct TPMS_TIME_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_TIME_INFO_mst;

typedef const struct TPMS_TIME_ATTEST_INFO_mst
{
```

```
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_TIME_ATTEST_INFO_mst;

typedef const struct TPMS_CERTIFY_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_CERTIFY_INFO_mst;

typedef const struct TPMS_QUOTE_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_QUOTE_INFO_mst;

typedef const struct TPMS_COMMAND_AUDIT_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[12];
} TPMS_COMMAND_AUDIT_INFO_mst;

typedef const struct TPMS_SESSION_AUDIT_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_SESSION_AUDIT_INFO_mst;

typedef const struct TPMS_CREATION_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_CREATION_INFO_mst;

typedef const struct TPMS_NV_CERTIFY_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMS_NV_CERTIFY_INFO_mst;

typedef const struct TPMS_NV_DIGEST_CERTIFY_INFO_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_NV_DIGEST_CERTIFY_INFO_mst;

typedef const struct TPMI_ST_ATTEST_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  ranges;
    UINT8  singles;
    UINT32 values[3];
} TPMI_ST_ATTEST_mst;

typedef struct TPMU_ATTEST_mst
{
```

```
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[8];
    UINT16 marshalingTypes[8];
} TPMU_ATTEST_mst;

typedef const struct TPMS_ATTEST_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[21];
} TPMS_ATTEST_mst;

typedef const struct TPMS_AUTH_COMMAND_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[12];
} TPMS_AUTH_COMMAND_mst;

typedef const struct TPMS_AUTH_RESPONSE_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMS_AUTH_RESPONSE_mst;

typedef const struct TPMI_AES_KEY_BITS_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[3];
} TPMI_AES_KEY_BITS_mst;

typedef const struct TPMI_SM4_KEY_BITS_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[1];
} TPMI_SM4_KEY_BITS_mst;

typedef const struct TPMI_CAMELLIA_KEY_BITS_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[3];
} TPMI_CAMELLIA_KEY_BITS_mst;

typedef struct TPMU_SYM_KEY_BITS_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[6];
    UINT16 marshalingTypes[6];
} TPMU_SYM_KEY_BITS_mst;

typedef struct TPMU_SYM_MODE_mst
{
```

```
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[6];
    UINT16 marshalingTypes[6];
} TPMU_SYM_MODE_mst;

typedef const struct TPMT_SYM_DEF_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMT_SYM_DEF_mst;

typedef const struct TPMT_SYM_DEF_OBJECT_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMT_SYM_DEF_OBJECT_mst;

typedef const struct TPMS_SYMCIPHER_PARMS_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[3];
} TPMS_SYMCIPHER_PARMS_mst;

typedef const struct TPMS_DERIVE_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_DERIVE_mst;

typedef const struct TPMS_SENSITIVE_CREATE_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_SENSITIVE_CREATE_mst;

typedef const struct TPMS_SCHEME_HASH_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[3];
} TPMS_SCHEME_HASH_mst;

typedef const struct TPMS_SCHEME_ECDAA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_SCHEME_ECDAA_mst;

typedef const struct TPMI_ALG_KEYEDHASH_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_KEYEDHASH_SCHEME_mst;

typedef const struct TPMS_SCHEME_XOR_mst
{
```

```
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_SCHEME_XOR_mst;

typedef struct TPMU_SCHEME_KEYEDHASH_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[3];
    UINT16 marshalingTypes[3];
} TPMU_SCHEME_KEYEDHASH_mst;

typedef const struct TPMT_KEYEDHASH_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_KEYEDHASH_SCHEME_mst;

typedef struct TPMU_SIG_SCHEME_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[8];
    UINT16 marshalingTypes[8];
} TPMU_SIG_SCHEME_mst;

typedef const struct TPMT_SIG_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_SIG_SCHEME_mst;

typedef struct TPMU_KDF_SCHEME_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[5];
    UINT16 marshalingTypes[5];
} TPMU_KDF_SCHEME_mst;

typedef const struct TPMT_KDF_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_KDF_SCHEME_mst;

typedef const struct TPMI_ALG_ASYM_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_ASYM_SCHEME_mst;

typedef struct TPMU_ASYM_SCHEME_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
```

```
    UINT32 selectors[11];
    UINT16 marshalingTypes[11];
} TPMU_ASYM_SCHEME_mst;

typedef const struct TPMI_ALG_RSA_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_RSA_SCHEME_mst;

typedef const struct TPMT_RSA_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_RSA_SCHEME_mst;

typedef const struct TPMI_ALG_RSA_DECRYPT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_RSA_DECRYPT_mst;

typedef const struct TPMT_RSA_DECRYPT_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_RSA_DECRYPT_mst;

typedef const struct TPMI_RSA_KEY_BITS_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[3];
} TPMI_RSA_KEY_BITS_mst;

typedef const struct TPMS_ECC_POINT_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_ECC_POINT_mst;

typedef const struct TPMI_ALG_ECC_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_ECC_SCHEME_mst;

typedef const struct TPMI_ECC_CURVE_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ECC_CURVE_mst;
```

```
typedef const struct TPMT_ECC_SCHEME_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_ECC_SCHEME_mst;

typedef const struct TPMS_ALGORITHM_DETAIL_ECC_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[33];
} TPMS_ALGORITHM_DETAIL_ECC_mst;

typedef const struct TPMS_SIGNATURE_RSA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_SIGNATURE_RSA_mst;

typedef const struct TPMS_SIGNATURE_ECC_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[9];
} TPMS_SIGNATURE_ECC_mst;

typedef struct TPMU_SIGNATURE_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[8];
    UINT16 marshalingTypes[8];
} TPMU_SIGNATURE_mst;

typedef const struct TPMT_SIGNATURE_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_SIGNATURE_mst;

typedef struct TPMU_ENCRYPTED_SECRET_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[4];
    UINT16 marshalingTypes[4];
} TPMU_ENCRYPTED_SECRET_mst;

typedef const struct TPMI_ALG_PUBLIC_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[4];
} TPMI_ALG_PUBLIC_mst;

typedef struct TPMU_PUBLIC_ID_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
```

```
    UINT32 selectors[4];
    UINT16 marshalingTypes[4];
} TPMU_PUBLIC_ID_mst;

typedef const struct TPMS_KEYEDHASH_PARMS_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[3];
} TPMS_KEYEDHASH_PARMS_mst;

typedef const struct TPMS_RSA_PARMS_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[12];
} TPMS_RSA_PARMS_mst;

typedef const struct TPMS_ECC_PARMS_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[12];
} TPMS_ECC_PARMS_mst;

typedef struct TPMU_PUBLIC_PARMS_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[4];
    UINT16 marshalingTypes[4];
} TPMU_PUBLIC_PARMS_mst;

typedef const struct TPMT_PUBLIC_PARMS_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_PUBLIC_PARMS_mst;

typedef const struct TPMT_PUBLIC_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[18];
} TPMT_PUBLIC_mst;

typedef struct TPMU_SENSITIVE_COMPOSITE_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[4];
    UINT16 marshalingTypes[4];
} TPMU_SENSITIVE_COMPOSITE_mst;

typedef const struct TPMT_SENSITIVE_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[12];
} TPMT_SENSITIVE_mst;

typedef const struct TPMS_NV_PIN_COUNTER_PARAMETERS_mst
{
```

```
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_NV_PIN_COUNTER_PARAMETERS_mst;

typedef const struct TPMS_NV_PUBLIC_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[15];
} TPMS_NV_PUBLIC_mst;

typedef const struct TPMS_NV_PUBLIC_EXP_ATTR_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[15];
} TPMS_NV_PUBLIC_EXP_ATTR_mst;

typedef struct TPMU_NV_PUBLIC_2_mst
{
    BYTE   countOfselectors;
    BYTE   modifiers;
    UINT16 offsetOfUnmarshalTypes;
    UINT32 selectors[3];
    UINT16 marshalingTypes[3];
} TPMU_NV_PUBLIC_2_mst;

typedef const struct TPMT_NV_PUBLIC_2_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMT_NV_PUBLIC_2_mst;

typedef const struct TPMS_CONTEXT_DATA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_CONTEXT_DATA_mst;

typedef const struct TPMS_CONTEXT_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[12];
} TPMS_CONTEXT_mst;

typedef const struct TPMS_CREATION_DATA_mst
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[21];
} TPMS_CREATION_DATA_mst;

typedef const struct TPM_AT_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[4];
} TPM_AT_mst;

typedef const struct TPMS_AC_OUTPUT_mst
```

```
{
    UINT8  marshalType;
    UINT8  elements;
    UINT16 values[6];
} TPMS_AC_OUTPUT_mst;

typedef const struct Type02_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type02_mst;

typedef const struct Type03_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type03_mst;

typedef const struct Type04_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type04_mst;

typedef const struct Type06_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type06_mst;

typedef const struct Type08_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type08_mst;

typedef const struct Type10_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[1];
} Type10_mst;

typedef const struct Type11_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[1];
} Type11_mst;

typedef const struct Type12_mst
{
```

```
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[2];
} Type12_mst;

typedef const struct Type13_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT8  entries;
    UINT32 values[1];
} Type13_mst;

typedef const struct Type15_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type15_mst;

typedef const struct Type17_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type17_mst;

typedef const struct Type18_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type18_mst;

typedef const struct Type19_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type19_mst;

typedef const struct Type20_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type20_mst;

typedef const struct Type22_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type22_mst;

typedef const struct Type23_mst
{
```

```
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type23_mst;

typedef const struct Type24_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type24_mst;

typedef const struct Type25_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type25_mst;

typedef const struct Type26_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type26_mst;

typedef const struct Type27_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type27_mst;

typedef const struct Type29_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type29_mst;

typedef const struct Type30_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type30_mst;

typedef const struct Type33_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type33_mst;

typedef const struct Type34_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
```

```
    UINT8  errorCode;
    UINT32 values[2];
} Type34_mst;

typedef const struct Type35_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type35_mst;

typedef const struct Type38_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type38_mst;

typedef const struct Type41_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type41_mst;

typedef const struct Type42_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type42_mst;

typedef const struct Type44_mst
{
    UINT8  marshalType;
    UINT8  modifiers;
    UINT8  errorCode;
    UINT32 values[2];
} Type44_mst;

// This structure combines all the individual marshaling structures to build
// something that can be referenced by offset rather than full address
typedef const struct MarshalData_st
{
    UintMarshal_mst                 UINT8_DATA;
    UintMarshal_mst                 UINT16_DATA;
    UintMarshal_mst                 UINT32_DATA;
    UintMarshal_mst                 UINT64_DATA;
    UintMarshal_mst                 INT8_DATA;
    UintMarshal_mst                 INT16_DATA;
    UintMarshal_mst                 INT32_DATA;
    UintMarshal_mst                 INT64_DATA;
    UintMarshal_mst                 UINT0_DATA;
    TPM_ECC_CURVE_mst               TPM_ECC_CURVE_DATA;
    TPM_CLOCK_ADJUST_mst            TPM_CLOCK_ADJUST_DATA;
    TPM_EO_mst                      TPM_EO_DATA;
    TPM_SU_mst                      TPM_SU_DATA;
    TPM_SE_mst                      TPM_SE_DATA;
    TPM_CAP_mst                     TPM_CAP_DATA;
    AttributesMarshal_mst           TPMA_ALGORITHM_DATA;
    AttributesMarshal_mst           TPMA_OBJECT_DATA;
    AttributesMarshal_mst           TPMA_SESSION_DATA;
```

```
AttributesMarshal_mst                TPMA_ACT_DATA;
TPMI_YES_NO_mst                      TPMI_YES_NO_DATA;
TPMI_DH_OBJECT_mst                   TPMI_DH_OBJECT_DATA;
TPMI_DH_PARENT_mst                   TPMI_DH_PARENT_DATA;
TPMI_DH_PERSISTENT_mst               TPMI_DH_PERSISTENT_DATA;
TPMI_DH_ENTITY_mst                   TPMI_DH_ENTITY_DATA;
TPMI_DH_PCR_mst                      TPMI_DH_PCR_DATA;
TPMI_SH_AUTH_SESSION_mst             TPMI_SH_AUTH_SESSION_DATA;
TPMI_SH_HMAC_mst                     TPMI_SH_HMAC_DATA;
TPMI_SH_POLICY_mst                   TPMI_SH_POLICY_DATA;
TPMI_DH_CONTEXT_mst                  TPMI_DH_CONTEXT_DATA;
TPMI_DH_SAVED_mst                    TPMI_DH_SAVED_DATA;
TPMI_RH_HIERARCHY_mst                TPMI_RH_HIERARCHY_DATA;
TPMI_RH_ENABLES_mst                  TPMI_RH_ENABLES_DATA;
TPMI_RH_HIERARCHY_AUTH_mst           TPMI_RH_HIERARCHY_AUTH_DATA;
TPMI_RH_HIERARCHY_POLICY_mst         TPMI_RH_HIERARCHY_POLICY_DATA;
TPMI_RH_BASE_HIERARCHY_mst           TPMI_RH_BASE_HIERARCHY_DATA;
TPMI_RH_PLATFORM_mst                 TPMI_RH_PLATFORM_DATA;
TPMI_RH_OWNER_mst                    TPMI_RH_OWNER_DATA;
TPMI_RH_ENDORSEMENT_mst              TPMI_RH_ENDORSEMENT_DATA;
TPMI_RH_PROVISION_mst                TPMI_RH_PROVISION_DATA;
TPMI_RH_CLEAR_mst                    TPMI_RH_CLEAR_DATA;
TPMI_RH_NV_AUTH_mst                  TPMI_RH_NV_AUTH_DATA;
TPMI_RH_LOCKOUT_mst                  TPMI_RH_LOCKOUT_DATA;
TPMI_RH_NV_INDEX_mst                 TPMI_RH_NV_INDEX_DATA;
TPMI_RH_NV_DEFINED_INDEX_mst         TPMI_RH_NV_DEFINED_INDEX_DATA;
TPMI_RH_LEGACY_NV_INDEX_mst          TPMI_RH_LEGACY_NV_INDEX_DATA;
TPMI_RH_NV_EXP_INDEX_mst             TPMI_RH_NV_EXP_INDEX_DATA;
TPMI_RH_AC_mst                       TPMI_RH_AC_DATA;
TPMI_RH_ACT_mst                      TPMI_RH_ACT_DATA;
TPMI_ALG_HASH_mst                    TPMI_ALG_HASH_DATA;
TPMI_ALG_ASYM_mst                    TPMI_ALG_ASYM_DATA;
TPMI_ALG_SYM_mst                     TPMI_ALG_SYM_DATA;
TPMI_ALG_SYM_OBJECT_mst              TPMI_ALG_SYM_OBJECT_DATA;
TPMI_ALG_SYM_MODE_mst                TPMI_ALG_SYM_MODE_DATA;
TPMI_ALG_KDF_mst                     TPMI_ALG_KDF_DATA;
TPMI_ALG_SIG_SCHEME_mst              TPMI_ALG_SIG_SCHEME_DATA;
TPMI_ECC_KEY_EXCHANGE_mst            TPMI_ECC_KEY_EXCHANGE_DATA;
TPMI_ST_COMMAND_TAG_mst              TPMI_ST_COMMAND_TAG_DATA;
TPMI_ALG_MAC_SCHEME_mst              TPMI_ALG_MAC_SCHEME_DATA;
TPMI_ALG_CIPHER_MODE_mst             TPMI_ALG_CIPHER_MODE_DATA;
TPMS_EMPTY_mst                       TPMS_EMPTY_DATA;
TPMS_ALGORITHM_DESCRIPTION_mst       TPMS_ALGORITHM_DESCRIPTION_DATA;
TPMU_HA_mst                          TPMU_HA_DATA;
TPMT_HA_mst                          TPMT_HA_DATA;
Tpm2bMarshal_mst                     TPM2B_DIGEST_DATA;
Tpm2bMarshal_mst                     TPM2B_DATA_DATA;
Tpm2bMarshal_mst                     TPM2B_EVENT_DATA;
Tpm2bMarshal_mst                     TPM2B_MAX_BUFFER_DATA;
Tpm2bMarshal_mst                     TPM2B_MAX_NV_BUFFER_DATA;
Tpm2bMarshal_mst                     TPM2B_TIMEOUT_DATA;
Tpm2bMarshal_mst                     TPM2B_IV_DATA;
NullUnionMarshal_mst                 NULL_UNION_DATA;
Tpm2bMarshal_mst                     TPM2B_NAME_DATA;
TPMS_PCR_SELECT_mst                  TPMS_PCR_SELECT_DATA;
TPMS_PCR_SELECTION_mst               TPMS_PCR_SELECTION_DATA;
TPMT_TK_CREATION_mst                 TPMT_TK_CREATION_DATA;
TPMT_TK_VERIFIED_mst                 TPMT_TK_VERIFIED_DATA;
TPMT_TK_AUTH_mst                     TPMT_TK_AUTH_DATA;
TPMT_TK_HASHCHECK_mst                TPMT_TK_HASHCHECK_DATA;
TPMS_ALG_PROPERTY_mst                TPMS_ALG_PROPERTY_DATA;
TPMS_TAGGED_PROPERTY_mst             TPMS_TAGGED_PROPERTY_DATA;
TPMS_TAGGED_PCR_SELECT_mst           TPMS_TAGGED_PCR_SELECT_DATA;
TPMS_TAGGED_POLICY_mst               TPMS_TAGGED_POLICY_DATA;
TPMS_ACT_DATA_mst                    TPMS_ACT_DATA_DATA;
ListMarshal_mst                      TPML_CC_DATA;
```

```
ListMarshal_mst                  TPML_CCA_DATA;
ListMarshal_mst                  TPML_ALG_DATA;
ListMarshal_mst                  TPML_HANDLE_DATA;
ListMarshal_mst                  TPML_DIGEST_DATA;
ListMarshal_mst                  TPML_DIGEST_VALUES_DATA;
ListMarshal_mst                  TPML_PCR_SELECTION_DATA;
ListMarshal_mst                  TPML_ALG_PROPERTY_DATA;
ListMarshal_mst                  TPML_TAGGED_TPM_PROPERTY_DATA;
ListMarshal_mst                  TPML_TAGGED_PCR_PROPERTY_DATA;
ListMarshal_mst                  TPML_ECC_CURVE_DATA;
ListMarshal_mst                  TPML_TAGGED_POLICY_DATA;
ListMarshal_mst                  TPML_ACT_DATA_DATA;
TPMU_CAPABILITIES_mst            TPMU_CAPABILITIES_DATA;
TPMS_CAPABILITY_DATA_mst         TPMS_CAPABILITY_DATA_DATA;
TPMU_SET_CAPABILITIES_mst        TPMU_SET_CAPABILITIES_DATA;
TPMS_SET_CAPABILITY_DATA_mst     TPMS_SET_CAPABILITY_DATA_DATA;
TPM2B_SET_CAPABILITY_DATA_mst    TPM2B_SET_CAPABILITY_DATA_DATA;
TPMS_CLOCK_INFO_mst              TPMS_CLOCK_INFO_DATA;
TPMS_TIME_INFO_mst               TPMS_TIME_INFO_DATA;
TPMS_TIME_ATTEST_INFO_mst        TPMS_TIME_ATTEST_INFO_DATA;
TPMS_CERTIFY_INFO_mst            TPMS_CERTIFY_INFO_DATA;
TPMS_QUOTE_INFO_mst              TPMS_QUOTE_INFO_DATA;
TPMS_COMMAND_AUDIT_INFO_mst      TPMS_COMMAND_AUDIT_INFO_DATA;
TPMS_SESSION_AUDIT_INFO_mst      TPMS_SESSION_AUDIT_INFO_DATA;
TPMS_CREATION_INFO_mst           TPMS_CREATION_INFO_DATA;
TPMS_NV_CERTIFY_INFO_mst         TPMS_NV_CERTIFY_INFO_DATA;
TPMS_NV_DIGEST_CERTIFY_INFO_mst  TPMS_NV_DIGEST_CERTIFY_INFO_DATA;
TPMI_ST_ATTEST_mst               TPMI_ST_ATTEST_DATA;
TPMU_ATTEST_mst                  TPMU_ATTEST_DATA;
TPMS_ATTEST_mst                  TPMS_ATTEST_DATA;
Tpm2bMarshal_mst                 TPM2B_ATTEST_DATA;
TPMS_AUTH_COMMAND_mst            TPMS_AUTH_COMMAND_DATA;
TPMS_AUTH_RESPONSE_mst           TPMS_AUTH_RESPONSE_DATA;
TPMI_AES_KEY_BITS_mst            TPMI_AES_KEY_BITS_DATA;
TPMI_SM4_KEY_BITS_mst            TPMI_SM4_KEY_BITS_DATA;
TPMI_CAMELLIA_KEY_BITS_mst       TPMI_CAMELLIA_KEY_BITS_DATA;
TPMU_SYM_KEY_BITS_mst            TPMU_SYM_KEY_BITS_DATA;
TPMU_SYM_MODE_mst                TPMU_SYM_MODE_DATA;
TPMT_SYM_DEF_mst                 TPMT_SYM_DEF_DATA;
TPMT_SYM_DEF_OBJECT_mst          TPMT_SYM_DEF_OBJECT_DATA;
Tpm2bMarshal_mst                 TPM2B_SYM_KEY_DATA;
TPMS_SYMCIPHER_PARMS_mst         TPMS_SYMCIPHER_PARMS_DATA;
Tpm2bMarshal_mst                 TPM2B_LABEL_DATA;
TPMS_DERIVE_mst                  TPMS_DERIVE_DATA;
Tpm2bMarshal_mst                 TPM2B_DERIVE_DATA;
Tpm2bMarshal_mst                 TPM2B_SENSITIVE_DATA_DATA;
TPMS_SENSITIVE_CREATE_mst        TPMS_SENSITIVE_CREATE_DATA;
Tpm2bsMarshal_mst                TPM2B_SENSITIVE_CREATE_DATA;
TPMS_SCHEME_HASH_mst             TPMS_SCHEME_HASH_DATA;
TPMS_SCHEME_ECDAA_mst            TPMS_SCHEME_ECDAA_DATA;
TPMI_ALG_KEYEDHASH_SCHEME_mst    TPMI_ALG_KEYEDHASH_SCHEME_DATA;
TPMS_SCHEME_XOR_mst              TPMS_SCHEME_XOR_DATA;
TPMU_SCHEME_KEYEDHASH_mst        TPMU_SCHEME_KEYEDHASH_DATA;
TPMT_KEYEDHASH_SCHEME_mst        TPMT_KEYEDHASH_SCHEME_DATA;
TPMU_SIG_SCHEME_mst              TPMU_SIG_SCHEME_DATA;
TPMT_SIG_SCHEME_mst              TPMT_SIG_SCHEME_DATA;
TPMU_KDF_SCHEME_mst              TPMU_KDF_SCHEME_DATA;
TPMT_KDF_SCHEME_mst              TPMT_KDF_SCHEME_DATA;
TPMI_ALG_ASYM_SCHEME_mst         TPMI_ALG_ASYM_SCHEME_DATA;
TPMU_ASYM_SCHEME_mst             TPMU_ASYM_SCHEME_DATA;
TPMI_ALG_RSA_SCHEME_mst          TPMI_ALG_RSA_SCHEME_DATA;
TPMT_RSA_SCHEME_mst              TPMT_RSA_SCHEME_DATA;
TPMI_ALG_RSA_DECRYPT_mst         TPMI_ALG_RSA_DECRYPT_DATA;
TPMT_RSA_DECRYPT_mst             TPMT_RSA_DECRYPT_DATA;
Tpm2bMarshal_mst                 TPM2B_PUBLIC_KEY_RSA_DATA;
TPMI_RSA_KEY_BITS_mst            TPMI_RSA_KEY_BITS_DATA;
```

```
Tpm2bMarshal_mst                    TPM2B_PRIVATE_KEY_RSA_DATA;
Tpm2bMarshal_mst                    TPM2B_ECC_PARAMETER_DATA;
TPMS_ECC_POINT_mst                  TPMS_ECC_POINT_DATA;
Tpm2bsMarshal_mst                   TPM2B_ECC_POINT_DATA;
TPMI_ALG_ECC_SCHEME_mst             TPMI_ALG_ECC_SCHEME_DATA;
TPMI_ECC_CURVE_mst                  TPMI_ECC_CURVE_DATA;
TPMT_ECC_SCHEME_mst                 TPMT_ECC_SCHEME_DATA;
TPMS_ALGORITHM_DETAIL_ECC_mst       TPMS_ALGORITHM_DETAIL_ECC_DATA;
TPMS_SIGNATURE_RSA_mst              TPMS_SIGNATURE_RSA_DATA;
TPMS_SIGNATURE_ECC_mst              TPMS_SIGNATURE_ECC_DATA;
TPMU_SIGNATURE_mst                  TPMU_SIGNATURE_DATA;
TPMT_SIGNATURE_mst                  TPMT_SIGNATURE_DATA;
TPMU_ENCRYPTED_SECRET_mst           TPMU_ENCRYPTED_SECRET_DATA;
Tpm2bMarshal_mst                    TPM2B_ENCRYPTED_SECRET_DATA;
TPMI_ALG_PUBLIC_mst                 TPMI_ALG_PUBLIC_DATA;
TPMU_PUBLIC_ID_mst                  TPMU_PUBLIC_ID_DATA;
TPMS_KEYEDHASH_PARMS_mst            TPMS_KEYEDHASH_PARMS_DATA;
TPMS_RSA_PARMS_mst                  TPMS_RSA_PARMS_DATA;
TPMS_ECC_PARMS_mst                  TPMS_ECC_PARMS_DATA;
TPMU_PUBLIC_PARMS_mst               TPMU_PUBLIC_PARMS_DATA;
TPMT_PUBLIC_PARMS_mst               TPMT_PUBLIC_PARMS_DATA;
TPMT_PUBLIC_mst                     TPMT_PUBLIC_DATA;
Tpm2bsMarshal_mst                   TPM2B_PUBLIC_DATA;
Tpm2bMarshal_mst                    TPM2B_TEMPLATE_DATA;
Tpm2bMarshal_mst                    TPM2B_PRIVATE_VENDOR_SPECIFIC_DATA;
TPMU_SENSITIVE_COMPOSITE_mst        TPMU_SENSITIVE_COMPOSITE_DATA;
TPMT_SENSITIVE_mst                  TPMT_SENSITIVE_DATA;
Tpm2bsMarshal_mst                   TPM2B_SENSITIVE_DATA;
Tpm2bMarshal_mst                    TPM2B_PRIVATE_DATA;
Tpm2bMarshal_mst                    TPM2B_ID_OBJECT_DATA;
TPMS_NV_PIN_COUNTER_PARAMETERS_mst  TPMS_NV_PIN_COUNTER_PARAMETERS_DATA;
AttributesMarshal_mst               TPMA_NV_DATA;
Attributes64Marshal_mst             TPMA_NV_EXP_DATA;
TPMS_NV_PUBLIC_mst                  TPMS_NV_PUBLIC_DATA;
Tpm2bsMarshal_mst                   TPM2B_NV_PUBLIC_DATA;
TPMS_NV_PUBLIC_EXP_ATTR_mst         TPMS_NV_PUBLIC_EXP_ATTR_DATA;
TPMU_NV_PUBLIC_2_mst                TPMU_NV_PUBLIC_2_DATA;
TPMT_NV_PUBLIC_2_mst                TPMT_NV_PUBLIC_2_DATA;
Tpm2bsMarshal_mst                   TPM2B_NV_PUBLIC_2_DATA;
Tpm2bMarshal_mst                    TPM2B_CONTEXT_SENSITIVE_DATA;
TPMS_CONTEXT_DATA_mst               TPMS_CONTEXT_DATA_DATA;
Tpm2bMarshal_mst                    TPM2B_CONTEXT_DATA_DATA;
TPMS_CONTEXT_mst                    TPMS_CONTEXT_DATA;
TPMS_CREATION_DATA_mst              TPMS_CREATION_DATA_DATA;
Tpm2bsMarshal_mst                   TPM2B_CREATION_DATA_DATA;
TPM_AT_mst                          TPM_AT_DATA;
TPMS_AC_OUTPUT_mst                  TPMS_AC_OUTPUT_DATA;
ListMarshal_mst                     TPML_AC_CAPABILITIES_DATA;
MinMaxMarshal_mst                   Type00_DATA;
MinMaxMarshal_mst                   Type01_DATA;
Type02_mst                          Type02_DATA;
Type03_mst                          Type03_DATA;
Type04_mst                          Type04_DATA;
MinMaxMarshal_mst                   Type05_DATA;
Type06_mst                          Type06_DATA;
MinMaxMarshal_mst                   Type07_DATA;
Type08_mst                          Type08_DATA;
Type10_mst                          Type10_DATA;
Type11_mst                          Type11_DATA;
Type12_mst                          Type12_DATA;
Type13_mst                          Type13_DATA;
Type15_mst                          Type15_DATA;
Type17_mst                          Type17_DATA;
Type18_mst                          Type18_DATA;
Type19_mst                          Type19_DATA;
Type20_mst                          Type20_DATA;
```

```
    Type22_mst                      Type22_DATA;
    Type23_mst                      Type23_DATA;
    Type24_mst                      Type24_DATA;
    Type25_mst                      Type25_DATA;
    Type26_mst                      Type26_DATA;
    Type27_mst                      Type27_DATA;
    MinMaxMarshal_mst               Type28_DATA;
    Type29_mst                      Type29_DATA;
    Type30_mst                      Type30_DATA;
    MinMaxMarshal_mst               Type31_DATA;
    MinMaxMarshal_mst               Type32_DATA;
    Type33_mst                      Type33_DATA;
    Type34_mst                      Type34_DATA;
    Type35_mst                      Type35_DATA;
    MinMaxMarshal_mst               Type36_DATA;
    MinMaxMarshal_mst               Type37_DATA;
    Type38_mst                      Type38_DATA;
    MinMaxMarshal_mst               Type39_DATA;
    MinMaxMarshal_mst               Type40_DATA;
    Type41_mst                      Type41_DATA;
    Type42_mst                      Type42_DATA;
    MinMaxMarshal_mst               Type43_DATA;
    Type44_mst                      Type44_DATA;
} MarshalData_st;

#endif  // _TABLE_MARSHAL_TYPES_H_
```

6.40 /tpm/include/private/Tpm.h

```
// Root header file for building any TPM.lib code

#ifndef _TPM_H_
#define _TPM_H_
// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/tpm_public.h>

#include "TpmAlgorithmDefines.h"
#include "LibSupport.h"         // Types from the library. These need to come before
                                // Global.h because some of the structures in
                                // that file depend on the structures used by the
                                // cryptographic libraries.
#include "GpMacros.h"           // Define additional macros
#include "Global.h"             // Define other TPM types
#include "InternalRoutines.h"   // Function prototypes

#endif  // _TPM_H_
```

6.41 /tpm/include/private/TpmASN1.h

```
//** Introduction
// This file contains the macro and structure definitions for the X509 commands and
// functions.

#ifndef _TPMASN1_H_
#define _TPMASN1_H_

//** Includes

#include "Tpm.h"
#include "OIDs.h"

//** Defined Constants
//*** ASN.1 Universal Types (Class 00b)
#define ASN1_EOC                0x00
```

```
#define ASN1_BOOLEAN            0x01
#define ASN1_INTEGER            0x02
#define ASN1_BITSTRING          0x03
#define ASN1_OCTET_STRING       0x04
#define ASN1_NULL               0x05
#define ASN1_OBJECT_IDENTIFIER  0x06
#define ASN1_OBJECT_DESCRIPTOR  0x07
#define ASN1_EXTERNAL           0x08
#define ASN1_REAL               0x09
#define ASN1_ENUMERATED         0x0A
#define ASN1_EMBEDDED           0x0B
#define ASN1_UTF8String         0x0C
#define ASN1_RELATIVE_OID       0x0D
#define ASN1_SEQUENCE           0x10  // Primitive + Constructed + 0x10
#define ASN1_SET                0x11  // Primitive + Constructed + 0x11
#define ASN1_NumericString      0x12
#define ASN1_PrintableString    0x13
#define ASN1_T61String          0x14
#define ASN1_VideoString        0x15
#define ASN1_IA5String          0x16
#define ASN1_UTCTime            0x17
#define ASN1_GeneralizeTime     0x18
#define ASN1_VisibleString      0x1A
#define ASN1_GeneralString      0x1B
#define ASN1_UniversalString    0x1C
#define ASN1_CHARACTER          STRING 0x1D
#define ASN1_BMPString          0x1E
#define ASN1_CONSTRUCTED        0x20

#define ASN1_APPLICAIION_SPECIFIC 0xA0

#define ASN1_CONSTRUCTED_SEQUENCE (ASN1_SEQUENCE + ASN1_CONSTRUCTED)

#define MAX_DEPTH 10  // maximum push depth for marshaling context.

//** Macros

//*** Unmarshaling Macros
#ifndef GOTO_ERROR_UNLESS
#  error missing GOTO_ERROR_UNLESS definition
#endif

// Checks the validity of the size making sure that there is no wrap around
#define CHECK_SIZE(context, length)                                         \
    GOTO_ERROR_UNLESS((((length) + (context)->offset) >= (context)->offset) \
                      && (((length) + (context)->offset) <= (context)->size))
#define NEXT_OCTET(context) ((context)->buffer[(context)->offset++])
#define PEEK_NEXT(context)  ((context)->buffer[(context)->offset])

//*** Marshaling Macros

// Marshaling works in reverse order. The offset is set to the top of the buffer and,
// as the buffer is filled, 'offset' counts down to zero. When the full thing is
// encoded it can be moved to the top of the buffer. This happens when the last
// context is closed.

#define CHECK_SPACE(context, length) GOTO_ERROR_UNLESS(context->offset > length)

//** Structures

typedef struct ASN1UnmarshalContext
{
    BYTE* buffer;  // pointer to the buffer
    INT16 size;    // size of the buffer (a negative number indicates
                   // a parsing failure).
    INT16 offset;  // current offset into the buffer (a negative number
```

```
                    // indicates a parsing failure). Not used
    BYTE tag;       // The last unmarshaled tag
} ASN1UnmarshalContext;

typedef struct ASN1MarshalContext
{
    BYTE* buffer;  // pointer to the start of the buffer
    INT16 offset;  // place on the top where the last entry was added
                   // items are added from the bottom up.
    INT16 end;     // the end offset of the current value
    INT16 depth;   // how many pushed end values.
    INT16 ends[MAX_DEPTH];
} ASN1MarshalContext;

#endif  // _TPMASN1_H_
```

6.42 /tpm/include/private/X509.h

```
//** Introduction
// This file contains the macro and structure definitions for the X509 commands and
// functions.

#ifndef _X509_H_
#define _X509_H_

//** Includes

#include "Tpm.h"
#include "TpmASN1.h"

//** Defined Constants

//*** X509 Application-specific types
#define X509_SELECTION          0xA0
#define X509_ISSUER_UNIQUE_ID   0xA1
#define X509_SUBJECT_UNIQUE_ID  0xA2
#define X509_EXTENSIONS         0xA3

// These defines give the order in which values appear in the TBScertificate
// of an x.509 certificate. These values are used to index into an array of
//
#define ENCODED_SIZE_REF        0
#define VERSION_REF             (ENCODED_SIZE_REF + 1)
#define SERIAL_NUMBER_REF       (VERSION_REF + 1)
#define SIGNATURE_REF           (SERIAL_NUMBER_REF + 1)
#define ISSUER_REF              (SIGNATURE_REF + 1)
#define VALIDITY_REF            (ISSUER_REF + 1)
#define SUBJECT_KEY_REF         (VALIDITY_REF + 1)
#define SUBJECT_PUBLIC_KEY_REF  (SUBJECT_KEY_REF + 1)
#define EXTENSIONS_REF          (SUBJECT_PUBLIC_KEY_REF + 1)
#define REF_COUNT               (EXTENSIONS_REF + 1)

//** Structures

// Used to access the fields of a TBSsignature some of which are in the in_CertifyX509
// structure and some of which are in the out_CertifyX509 structure.
typedef struct stringRef
{
    BYTE* buf;
    INT16 len;
} stringRef;

// This is defined to avoid bit by bit comparisons within a UINT32
typedef union x509KeyUsageUnion
{
```

```
    TPMA_X509_KEY_USAGE x509;
    UINT32              integer;
} x509KeyUsageUnion;

//** Global X509 Constants
// These values are instanced by X509_spt.c and referenced by other X509-related
// files.

// This is the DER-encoded value for the Key Usage OID  (2.5.29.15). This is the
// full OID, not just the numeric value
#define OID_KEY_USAGE_EXTENSION_VALUE 0x06, 0x03, 0x55, 0x1D, 0x0F
MAKE_OID(_KEY_USAGE_EXTENSION);

// This is the DER-encoded value for the TCG-defined TPMA_OBJECT OID
// (2.23.133.10.1.1.1)
#define OID_TCG_TPMA_OBJECT_VALUE 0x06, 0x07, 0x67, 0x81, 0x05, 0x0a, 0x01, 0x01, 0x01
MAKE_OID(_TCG_TPMA_OBJECT);

#ifdef _X509_SPT_
// If a bit is SET in KEY_USAGE_SIGN is also SET in keyUsage then
// the associated key has to have 'sign' SET.
const x509KeyUsageUnion KEY_USAGE_SIGN = {TPMA_X509_KEY_USAGE_INITIALIZER(
    /* bits_at_0        */ 0,
    /* decipheronly     */ 0,
    /* encipheronly     */ 0,
    /* crlsign          */ 1,
    /* keycertsign      */ 1,
    /* keyagreement     */ 0,
    /* dataencipherment */ 0,
    /* keyencipherment */ 0,
    /* nonrepudiation */ 0,
    /* digitalsignature */ 1)};
// If a bit is SET in KEY_USAGE_DECRYPT is also SET in keyUsage then
// the associated key has to have 'decrypt' SET.
const x509KeyUsageUnion KEY_USAGE_DECRYPT = {TPMA_X509_KEY_USAGE_INITIALIZER(
    /* bits_at_0        */ 0,
    /* decipheronly     */ 1,
    /* encipheronly     */ 1,
    /* crlsign          */ 0,
    /* keycertsign      */ 0,
    /* keyagreement     */ 1,
    /* dataencipherment */ 1,
    /* keyencipherment */ 1,
    /* nonrepudiation */ 0,
    /* digitalsignature */ 0)};
#else
extern x509KeyUsageUnion KEY_USAGE_SIGN;
extern x509KeyUsageUnion KEY_USAGE_DECRYPT;
#endif

#endif  // _X509_H_
```

6.43 /tpm/include/private/prototypes/ActivateCredential_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ActivateCredential  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ACTIVATECREDENTIAL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ACTIVATECREDENTIAL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT         activateHandle;
```

```
    TPMI_DH_OBJECT          keyHandle;
    TPM2B_ID_OBJECT         credentialBlob;
    TPM2B_ENCRYPTED_SECRET secret;
} ActivateCredential_In;

// Output structure definition
typedef struct
{
    TPM2B_DIGEST certInfo;
} ActivateCredential_Out;

// Response code modifiers
#    define RC_ActivateCredential_activateHandle (TPM_RC_H + TPM_RC_1)
#    define RC_ActivateCredential_keyHandle      (TPM_RC_H + TPM_RC_2)
#    define RC_ActivateCredential_credentialBlob (TPM_RC_P + TPM_RC_1)
#    define RC_ActivateCredential_secret         (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_ActivateCredential(ActivateCredential_In* in, ActivateCredential_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ACTIVATECREDENTIAL_FP_H_
#endif    // CC_ActivateCredential
```

6.44 /tpm/include/private/prototypes/ACT_SetTimeout_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ACT_SetTimeout  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ACT_SETTIMEOUT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ACT_SETTIMEOUT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_ACT actHandle;
    UINT32      startTimeout;
} ACT_SetTimeout_In;

// Response code modifiers
#    define RC_ACT_SetTimeout_actHandle    (TPM_RC_H + TPM_RC_1)
#    define RC_ACT_SetTimeout_startTimeout (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ACT_SetTimeout(ACT_SetTimeout_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ACT_SETTIMEOUT_FP_H_
#endif    // CC_ACT_SetTimeout
```

6.45 /tpm/include/private/prototypes/ACT_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes 1.00
 *  Date: Oct 24, 2019  Time: 10:38:43AM
 */

#ifndef _ACT_SPT_FP_H_
#define _ACT_SPT_FP_H_

//*** ActStartup()
// This function is called by TPM2_Startup() to initialize the ACT counter values.
BOOL ActStartup(STARTUP_TYPE type);
```

```
//*** ActGetSignaled()
// This function returns the state of the signaled flag associated with an ACT.
BOOL ActGetSignaled(TPM_RH actHandle);

//***ActShutdown()
// This function saves the current state of the counters
BOOL ActShutdown(TPM_SU state  //IN: the type of the shutdown.
);

//*** ActIsImplemented()
// This function determines if an ACT is implemented in both the TPM and the platform
// code.
BOOL ActIsImplemented(UINT32 act);

//***ActCounterUpdate()
// This function updates the ACT counter. If the counter already has a pending update,
// it returns TPM_RC_RETRY so that the update can be tried again later.
TPM_RC
ActCounterUpdate(TPM_RH handle,   //IN: the handle of the act
                 UINT32 newValue  //IN: the value to set in the ACT
);

//*** ActGetCapabilityData()
// This function returns the list of ACT data
//  Return Type: TPMI_YES_NO
//      YES             if more ACT data is available
//      NO              if no more ACT data to
TPMI_YES_NO
ActGetCapabilityData(TPM_HANDLE     actHandle,  // IN: the handle for the starting ACT
                     UINT32         maxCount,   // IN: maximum allowed return values
                     TPML_ACT_DATA* actList     // OUT: ACT data list
);

//*** ActGetOneCapability()
// This function returns an ACT's capability, if present.
BOOL ActGetOneCapability(TPM_HANDLE     actHandle,  // IN: the handle for the ACT
                         TPMS_ACT_DATA* actData     // OUT: ACT data
);

#endif  // _ACT_SPT_FP_H_
```

6.46 /tpm/include/private/prototypes/AC_GetCapability_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_AC_GetCapability  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_AC_GETCAPABILITY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_AC_GETCAPABILITY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_AC ac;
    TPM_AT     capability;
    UINT32     count;
} AC_GetCapability_In;

// Output structure definition
typedef struct
{
    TPMI_YES_NO          moreData;
    TPML_AC_CAPABILITIES capabilitiesData;
} AC_GetCapability_Out;
```

```
// Response code modifiers
#    define RC_AC_GetCapability_ac          (TPM_RC_H + TPM_RC_1)
#    define RC_AC_GetCapability_capability (TPM_RC_P + TPM_RC_1)
#    define RC_AC_GetCapability_count       (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_AC_GetCapability(AC_GetCapability_In* in, AC_GetCapability_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_AC_GETCAPABILITY_FP_H_
#endif    // CC_AC_GetCapability
```

6.47 /tpm/include/private/prototypes/AC_Send_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_AC_Send  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_AC_SEND_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_AC_SEND_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   sendObject;
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_AC       ac;
    TPM2B_MAX_BUFFER acDataIn;
} AC_Send_In;

// Output structure definition
typedef struct
{
    TPMS_AC_OUTPUT acDataOut;
} AC_Send_Out;

// Response code modifiers
#    define RC_AC_Send_sendObject (TPM_RC_H + TPM_RC_1)
#    define RC_AC_Send_authHandle (TPM_RC_H + TPM_RC_2)
#    define RC_AC_Send_ac         (TPM_RC_H + TPM_RC_3)
#    define RC_AC_Send_acDataIn   (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_AC_Send(AC_Send_In* in, AC_Send_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_AC_SEND_FP_H_
#endif    // CC_AC_Send
```

6.48 /tpm/include/private/prototypes/AC_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _AC_SPT_FP_H_
#define _AC_SPT_FP_H_

//*** AcToCapabilities()
// This function returns a pointer to a list of AC capabilities.
TPML_AC_CAPABILITIES* AcToCapabilities(TPMI_RH_AC component  // IN: component
);
```

```
//*** AcIsAccessible()
// Function to determine if an AC handle references an actual AC
//  Return Type: BOOL
BOOL AcIsAccessible(TPM_HANDLE acHandle);

//*** AcCapabilitiesGet()
// This function returns a list of capabilities associated with an AC
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
AcCapabilitiesGet(TPMI_RH_AC            component,      // IN: the component
                  TPM_AT                type,           // IN: start capability type
                  UINT32                count,          // IN: requested number
                  TPML_AC_CAPABILITIES* capabilityList  // OUT: list of handle
);

//*** AcSendObject()
// Stub to handle sending of an AC object
//  Return Type: TPM_RC
TPM_RC
AcSendObject(TPM_HANDLE      acHandle,  // IN: Handle of AC receiving object
             OBJECT*         object,    // IN: object structure to send
             TPMS_AC_OUTPUT* acDataOut  // OUT: results of operation
);

#endif  // _AC_SPT_FP_H_
```

6.49 /tpm/include/private/prototypes/AlgorithmCap_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _ALGORITHM_CAP_FP_H_
#define _ALGORITHM_CAP_FP_H_

//** AlgorithmCapGetImplemented()
// This function is used by TPM2_GetCapability() to return a list of the
// implemented algorithms.
//
//  Return Type: TPMI_YES_NO
//  YES         more algorithms to report
//  NO          no more algorithms to report
TPMI_YES_NO
AlgorithmCapGetImplemented(TPM_ALG_ID algID,  // IN: the starting algorithm ID
                           UINT32     count,  // IN: count of returned algorithms
                           TPML_ALG_PROPERTY* algList  // OUT: algorithm list
);

//** AlgorithmCapGetOneImplemented()
// This function returns whether a single algorithm was implemented, along
// with its properties (if implemented).
BOOL AlgorithmCapGetOneImplemented(
    TPM_ALG_ID         algID,       // IN: the algorithm ID
    TPMS_ALG_PROPERTY* algProperty  // OUT: algorithm properties
);

//** AlgorithmGetImplementedVector()
// This function returns the bit vector of the implemented algorithms.
LIB_EXPORT
void AlgorithmGetImplementedVector(
    ALGORITHM_VECTOR* implemented  // OUT: the implemented bits are SET
```

```
);

#endif  // _ALGORITHM_CAP_FP_H_
```

6.50 /tpm/include/private/prototypes/AlgorithmTests_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _ALGORITHM_TESTS_FP_H_
#define _ALGORITHM_TESTS_FP_H_

#if ENABLE_SELF_TESTS

//*** TestAlgorithm()
// Dispatches to the correct test function for the algorithm or gets a list of
// testable algorithms.
//
// If 'toTest' is not NULL, then the test decisions are based on the algorithm
// selections in 'toTest'. Otherwise, 'g_toTest' is used. When bits are clear in
// 'g_toTest' they will also be cleared 'toTest'.
//
// If there doesn't happen to be a test for the algorithm, its associated bit is
// quietly cleared.
//
// If 'alg' is zero (TPM_ALG_ERROR), then the toTest vector is cleared of any bits
// for which there is no test (i.e. no tests are actually run but the vector is
// cleared).
//
// Note: 'toTest' will only ever have bits set for implemented algorithms but 'alg'
// can be anything.
//  Return Type: TPM_RC
//      TPM_RC_CANCELED     test was canceled
LIB_EXPORT
TPM_RC
TestAlgorithm(TPM_ALG_ID alg, ALGORITHM_VECTOR* toTest);
#endif  // ENABLE_SELF_TESTS

#endif  // _ALGORITHM_TESTS_FP_H_
```

6.51 /tpm/include/private/prototypes/Attest_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _ATTEST_SPT_FP_H_
#define _ATTEST_SPT_FP_H_

//***FillInAttestInfo()
// Fill in common fields of TPMS_ATTEST structure.
void FillInAttestInfo(
    TPMI_DH_OBJECT   signHandle,  // IN: handle of signing object
    TPMT_SIG_SCHEME* scheme,      // IN/OUT: scheme to be used for signing
    TPM2B_DATA*      data,        // IN: qualifying data
    TPMS_ATTEST*     attest       // OUT: attest structure
);

//***SignAttestInfo()
// Sign a TPMS_ATTEST structure. If signHandle is TPM_RH_NULL, a null signature
// is returned.
```

```
//
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   'signHandle' references not a signing key
//      TPM_RC_SCHEME       'scheme' is not compatible with 'signHandle' type
//      TPM_RC_VALUE        digest generated for the given 'scheme' is greater than
//                          the modulus of 'signHandle' (for an RSA key);
//                          invalid commit status or failed to generate "r" value
//                          (for an ECC key)
TPM_RC
SignAttestInfo(OBJECT*          signKey,         // IN: sign object
               TPMT_SIG_SCHEME* scheme,          // IN: sign scheme
               TPMS_ATTEST*     certifyInfo,     // IN: the data to be signed
               TPM2B_DATA*      qualifyingData,  // IN: extra data for the signing
                                                 //     process
               TPM2B_ATTEST* attest,             // OUT: marshaled attest blob to be
                                                 //     signed
               TPMT_SIGNATURE* signature         // OUT: signature
);

//*** IsSigningObject()
// Checks to see if the object is OK for signing. This is here rather than in
// Object_spt.c because all the attestation commands use this file but not
// Object_spt.c.
//  Return Type: BOOL
//      TRUE(1)         object may sign
//      FALSE(0)        object may not sign
BOOL IsSigningObject(OBJECT* object  // IN:
);

#endif  // _ATTEST_SPT_FP_H_
```

6.52 /tpm/include/private/prototypes/Bits_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _BITS_FP_H_
#define _BITS_FP_H_

//*** TestBit()
// This function is used to check the setting of a bit in an array of bits.
//  Return Type: BOOL
//      TRUE(1)         bit is set
//      FALSE(0)        bit is not set
BOOL TestBit(unsigned int bitNum,       // IN: number of the bit in 'bArray'
             BYTE*        bArray,       // IN: array containing the bits
             unsigned int bytesInArray  // IN: size in bytes of 'bArray'
);

//*** SetBit()
// This function will set the indicated bit in 'bArray'.
void SetBit(unsigned int bitNum,       // IN: number of the bit in 'bArray'
            BYTE*        bArray,       // IN: array containing the bits
            unsigned int bytesInArray  // IN: size in bytes of 'bArray'
);

//*** ClearBit()
// This function will clear the indicated bit in 'bArray'.
void ClearBit(unsigned int bitNum,       // IN: number of the bit in 'bArray'.
              BYTE*        bArray,       // IN: array containing the bits
              unsigned int bytesInArray  // IN: size in bytes of 'bArray'
);
```

```
#endif  // _BITS_FP_H_
```

6.53 /tpm/include/private/prototypes/CertifyCreation_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_CertifyCreation  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFYCREATION_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFYCREATION_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   signHandle;
    TPMI_DH_OBJECT   objectHandle;
    TPM2B_DATA       qualifyingData;
    TPM2B_DIGEST     creationHash;
    TPMT_SIG_SCHEME  inScheme;
    TPMT_TK_CREATION creationTicket;
} CertifyCreation_In;

// Output structure definition
typedef struct
{
    TPM2B_ATTEST   certifyInfo;
    TPMT_SIGNATURE signature;
} CertifyCreation_Out;

// Response code modifiers
#    define RC_CertifyCreation_signHandle     (TPM_RC_H + TPM_RC_1)
#    define RC_CertifyCreation_objectHandle   (TPM_RC_H + TPM_RC_2)
#    define RC_CertifyCreation_qualifyingData (TPM_RC_P + TPM_RC_1)
#    define RC_CertifyCreation_creationHash   (TPM_RC_P + TPM_RC_2)
#    define RC_CertifyCreation_inScheme       (TPM_RC_P + TPM_RC_3)
#    define RC_CertifyCreation_creationTicket (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_CertifyCreation(CertifyCreation_In* in, CertifyCreation_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFYCREATION_FP_H_
#endif    // CC_CertifyCreation
```

6.54 /tpm/include/private/prototypes/CertifyX509_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_CertifyX509  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFYX509_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFYX509_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   objectHandle;
    TPMI_DH_OBJECT   signHandle;
    TPM2B_DATA       reserved;
    TPMT_SIG_SCHEME  inScheme;
    TPM2B_MAX_BUFFER partialCertificate;
} CertifyX509_In;

// Output structure definition
```

```
typedef struct
{
    TPM2B_MAX_BUFFER addedToCertificate;
    TPM2B_DIGEST     tbsDigest;
    TPMT_SIGNATURE   signature;
} CertifyX509_Out;

// Response code modifiers
#    define RC_CertifyX509_objectHandle       (TPM_RC_H + TPM_RC_1)
#    define RC_CertifyX509_signHandle         (TPM_RC_H + TPM_RC_2)
#    define RC_CertifyX509_reserved           (TPM_RC_P + TPM_RC_1)
#    define RC_CertifyX509_inScheme           (TPM_RC_P + TPM_RC_2)
#    define RC_CertifyX509_partialCertificate (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_CertifyX509(CertifyX509_In* in, CertifyX509_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFYX509_FP_H_
#endif    // CC_CertifyX509
```

6.55 /tpm/include/private/prototypes/Certify_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Certify  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT  objectHandle;
    TPMI_DH_OBJECT  signHandle;
    TPM2B_DATA      qualifyingData;
    TPMT_SIG_SCHEME inScheme;
} Certify_In;

// Output structure definition
typedef struct
{
    TPM2B_ATTEST   certifyInfo;
    TPMT_SIGNATURE signature;
} Certify_Out;

// Response code modifiers
#    define RC_Certify_objectHandle   (TPM_RC_H + TPM_RC_1)
#    define RC_Certify_signHandle     (TPM_RC_H + TPM_RC_2)
#    define RC_Certify_qualifyingData (TPM_RC_P + TPM_RC_1)
#    define RC_Certify_inScheme       (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_Certify(Certify_In* in, Certify_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CERTIFY_FP_H_
#endif    // CC_Certify
```

6.56 /tpm/include/private/prototypes/ChangeEPS_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ChangeEPS  // Command must be enabled
```

```
#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CHANGEEPS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CHANGEEPS_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PLATFORM authHandle;
} ChangeEPS_In;

// Response code modifiers
#    define RC_ChangeEPS_authHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ChangeEPS(ChangeEPS_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CHANGEEPS_FP_H_
#endif    // CC_ChangeEPS
```

6.57 /tpm/include/private/prototypes/ChangePPS_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ChangePPS  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CHANGEPPS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CHANGEPPS_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PLATFORM authHandle;
} ChangePPS_In;

// Response code modifiers
#    define RC_ChangePPS_authHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ChangePPS(ChangePPS_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CHANGEPPS_FP_H_
#endif    // CC_ChangePPS
```

6.58 /tpm/include/private/prototypes/ClearControl_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ClearControl  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLEARCONTROL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLEARCONTROL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_CLEAR auth;
    TPMI_YES_NO   disable;
} ClearControl_In;

// Response code modifiers
#    define RC_ClearControl_auth    (TPM_RC_H + TPM_RC_1)
#    define RC_ClearControl_disable (TPM_RC_P + TPM_RC_1)
```

```
// Function prototype
TPM_RC
TPM2_ClearControl(ClearControl_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLEARCONTROL_FP_H_
#endif    // CC_ClearControl
```

6.59 /tpm/include/private/prototypes/Clear_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Clear  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLEAR_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLEAR_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_CLEAR authHandle;
} Clear_In;

// Response code modifiers
#    define RC_Clear_authHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_Clear(Clear_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLEAR_FP_H_
#endif    // CC_Clear
```

6.60 /tpm/include/private/prototypes/ClockRateAdjust_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ClockRateAdjust  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLOCKRATEADJUST_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLOCKRATEADJUST_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION auth;
    TPM_CLOCK_ADJUST  rateAdjust;
} ClockRateAdjust_In;

// Response code modifiers
#    define RC_ClockRateAdjust_auth       (TPM_RC_H + TPM_RC_1)
#    define RC_ClockRateAdjust_rateAdjust (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ClockRateAdjust(ClockRateAdjust_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLOCKRATEADJUST_FP_H_
#endif    // CC_ClockRateAdjust
```

6.61 /tpm/include/private/prototypes/ClockSet_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT
```

```
#if CC_ClockSet  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLOCKSET_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLOCKSET_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION auth;
    UINT64            newTime;
} ClockSet_In;

// Response code modifiers
#    define RC_ClockSet_auth    (TPM_RC_H + TPM_RC_1)
#    define RC_ClockSet_newTime (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ClockSet(ClockSet_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CLOCKSET_FP_H_
#endif    // CC_ClockSet
```

6.62 /tpm/include/private/prototypes/CommandAudit_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 04:23:27PM
 */

#ifndef _COMMAND_AUDIT_FP_H_
#define _COMMAND_AUDIT_FP_H_

//*** CommandAuditPreInstall_Init()
// This function initializes the command audit list. This function simulates
// the behavior of manufacturing. A function is used instead of a structure
// definition because this is easier than figuring out the initialization value
// for a bit array.
//
// This function would not be implemented outside of a manufacturing or
// simulation environment.
void CommandAuditPreInstall_Init(void);

//*** CommandAuditStartup()
// This function clears the command audit digest on a TPM Reset.
BOOL CommandAuditStartup(STARTUP_TYPE type  // IN: start up type
);

//*** CommandAuditSet()
// This function will SET the audit flag for a command. This function
// will not SET the audit flag for a command that is not implemented. This
// ensures that the audit status is not SET when TPM2_GetCapability() is
// used to read the list of audited commands.
//
// This function is only used by TPM2_SetCommandCodeAuditStatus().
//
// The actions in TPM2_SetCommandCodeAuditStatus() are expected to cause the
// changes to be saved to NV after it is setting and clearing bits.
//  Return Type: BOOL
//      TRUE(1)         command code audit status was changed
//      FALSE(0)        command code audit status was not changed
BOOL CommandAuditSet(TPM_CC commandCode  // IN: command code
);
```

```
//*** CommandAuditClear()
// This function will CLEAR the audit flag for a command. It will not CLEAR the
// audit flag for TPM_CC_SetCommandCodeAuditStatus().
//
// This function is only used by TPM2_SetCommandCodeAuditStatus().
//
// The actions in TPM2_SetCommandCodeAuditStatus() are expected to cause the
// changes to be saved to NV after it is setting and clearing bits.
//  Return Type: BOOL
//      TRUE(1)         command code audit status was changed
//      FALSE(0)        command code audit status was not changed
BOOL CommandAuditClear(TPM_CC commandCode  // IN: command code
);

//*** CommandAuditIsRequired()
// This function indicates if the audit flag is SET for a command.
//  Return Type: BOOL
//      TRUE(1)         command is audited
//      FALSE(0)        command is not audited
BOOL CommandAuditIsRequired(COMMAND_INDEX commandIndex  // IN: command index
);

//*** CommandAuditCapGetCCList()
// This function returns a list of commands that have their audit bit SET.
//
// The list starts at the input commandCode.
//  Return Type: TPMI_YES_NO
//      YES         if there are more command code available
//      NO          all the available command code has been returned
TPMI_YES_NO
CommandAuditCapGetCCList(TPM_CC   commandCode,  // IN: start command code
                         UINT32   count,        // IN: count of returned TPM_CC
                         TPML_CC* commandList   // OUT: list of TPM_CC
);

//*** CommandAuditCapGetOneCC()
// This function returns true if a command has its audit bit set.
BOOL CommandAuditCapGetOneCC(TPM_CC commandCode  // IN: command code
);

//*** CommandAuditGetDigest
// This command is used to create a digest of the commands being audited. The
// commands are processed in ascending numeric order with a list of TPM_CC being
// added to a hash. This operates as if all the audited command codes were
// concatenated and then hashed.
void CommandAuditGetDigest(TPM2B_DIGEST* digest  // OUT: command digest
);

#endif  // _COMMAND_AUDIT_FP_H_
```

6.63 /tpm/include/private/prototypes/CommandCodeAttributes_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _COMMAND_CODE_ATTRIBUTES_FP_H_
#define _COMMAND_CODE_ATTRIBUTES_FP_H_

//*** GetClosestCommandIndex()
// This function returns the command index for the command with a value that is
// equal to or greater than the input value
//  Return Type: COMMAND_INDEX
//  UNIMPLEMENTED_COMMAND_INDEX     command is not implemented
```

```
//  other                           index of a command
COMMAND_INDEX
GetClosestCommandIndex(TPM_CC commandCode  // IN: the command code to start at
);

//*** CommandCodeToComandIndex()
// This function returns the index in the various attributes arrays of the
// command.
//  Return Type: COMMAND_INDEX
//  UNIMPLEMENTED_COMMAND_INDEX     command is not implemented
//  other                           index of the command
COMMAND_INDEX
CommandCodeToCommandIndex(TPM_CC commandCode  // IN: the command code to look up
);

//*** GetNextCommandIndex()
// This function returns the index of the next implemented command.
//  Return Type: COMMAND_INDEX
//  UNIMPLEMENTED_COMMAND_INDEX     no more implemented commands
//  other                           the index of the next implemented command
COMMAND_INDEX
GetNextCommandIndex(COMMAND_INDEX commandIndex  // IN: the starting index
);

//*** GetCommandCode()
// This function returns the commandCode associated with the command index
TPM_CC
GetCommandCode(COMMAND_INDEX commandIndex  // IN: the command index
);

//*** CommandAuthRole()
//
//  This function returns the authorization role required of a handle.
//
//  Return Type: AUTH_ROLE
//  AUTH_NONE       no authorization is required
//  AUTH_USER       user role authorization is required
//  AUTH_ADMIN      admin role authorization is required
//  AUTH_DUP        duplication role authorization is required
AUTH_ROLE
CommandAuthRole(COMMAND_INDEX commandIndex,  // IN: command index
                UINT32        handleIndex    // IN: handle index (zero based)
);

//*** EncryptSize()
// This function returns the size of the decrypt size field. This function returns
// 0 if encryption is not allowed
//  Return Type: int
//  0       encryption not allowed
//  2       size field is two bytes
//  4       size field is four bytes
int EncryptSize(COMMAND_INDEX commandIndex  // IN: command index
);

//*** DecryptSize()
// This function returns the size of the decrypt size field. This function returns
// 0 if decryption is not allowed
//  Return Type: int
//  0       encryption not allowed
//  2       size field is two bytes
//  4       size field is four bytes
int DecryptSize(COMMAND_INDEX commandIndex  // IN: command index
);

//*** IsSessionAllowed()
//
```

```
// This function indicates if the command is allowed to have sessions.
//
// This function must not be called if the command is not known to be implemented.
//
//  Return Type: BOOL
//      TRUE(1)         session is allowed with this command
//      FALSE(0)        session is not allowed with this command
BOOL IsSessionAllowed(COMMAND_INDEX commandIndex  // IN: the command to be checked
);

//*** IsHandleInResponse()
// This function determines if a command has a handle in the response
BOOL IsHandleInResponse(COMMAND_INDEX commandIndex);

//*** IsWriteOperation()
// Checks to see if an operation will write to an NV Index and is subject to being
// blocked by read-lock
BOOL IsWriteOperation(COMMAND_INDEX commandIndex  // IN: Command to check
);

//*** IsReadOperation()
// Checks to see if an operation will write to an NV Index and is
// subject to being blocked by write-lock.
BOOL IsReadOperation(COMMAND_INDEX commandIndex  // IN: Command to check
);

//*** CommandCapGetCCList()
// This function returns a list of implemented commands and command attributes
// starting from the command in 'commandCode'.
//  Return Type: TPMI_YES_NO
//      YES         more command attributes are available
//      NO          no more command attributes are available
TPMI_YES_NO
CommandCapGetCCList(TPM_CC commandCode,  // IN: start command code
                    UINT32 count,        // IN: maximum count for number of entries in
                                         //     'commandList'
                    TPML_CCA* commandList  // OUT: list of TPMA_CC
);

//*** CommandCapGetOneCC()
// This function checks whether a command is implemented, and returns its
// attributes if so.
BOOL CommandCapGetOneCC(TPM_CC   commandCode,       // IN: command code
                        TPMA_CC* commandAttributes  // OUT: Command attributes
);

//*** IsVendorCommand()
// Function indicates if a command index references a vendor command.
//  Return Type: BOOL
//      TRUE(1)         command is a vendor command
//      FALSE(0)        command is not a vendor command
BOOL IsVendorCommand(COMMAND_INDEX commandIndex  // IN: command index to check
);

#endif  // _COMMAND_CODE_ATTRIBUTES_FP_H_
```

6.64 /tpm/include/private/prototypes/CommandDispatcher_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  7, 2020  Time: 07:06:44PM
 */

#ifndef _COMMAND_DISPATCHER_FP_H_
#define _COMMAND_DISPATCHER_FP_H_
```

```
//** ParseHandleBuffer()
// This is the table-driven version of the handle buffer unmarshaling code
TPM_RC
ParseHandleBuffer(COMMAND* command);

//** CommandDispatcher()
// Function to unmarshal the command parameters, call the selected action code, and
// marshal the response parameters.
TPM_RC
CommandDispatcher(COMMAND* command);

#endif  // _COMMAND_DISPATCHER_FP_H_
```

6.65 /tpm/include/private/prototypes/Commit_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Commit  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_COMMIT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_COMMIT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT       signHandle;
    TPM2B_ECC_POINT      P1;
    TPM2B_SENSITIVE_DATA s2;
    TPM2B_ECC_PARAMETER  y2;
} Commit_In;

// Output structure definition
typedef struct
{
    TPM2B_ECC_POINT K;
    TPM2B_ECC_POINT L;
    TPM2B_ECC_POINT E;
    UINT16          counter;
} Commit_Out;

// Response code modifiers
#    define RC_Commit_signHandle (TPM_RC_H + TPM_RC_1)
#    define RC_Commit_P1         (TPM_RC_P + TPM_RC_1)
#    define RC_Commit_s2         (TPM_RC_P + TPM_RC_2)
#    define RC_Commit_y2         (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_Commit(Commit_In* in, Commit_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_COMMIT_FP_H_
#endif    // CC_Commit
```

6.66 /tpm/include/private/prototypes/ContextLoad_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ContextLoad  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CONTEXTLOAD_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CONTEXTLOAD_FP_H_

// Input structure definition
```

```
typedef struct
{
    TPMS_CONTEXT context;
} ContextLoad_In;

// Output structure definition
typedef struct
{
    TPMI_DH_CONTEXT loadedHandle;
} ContextLoad_Out;

// Response code modifiers
#    define RC_ContextLoad_context (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ContextLoad(ContextLoad_In* in, ContextLoad_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CONTEXTLOAD_FP_H_
#endif    // CC_ContextLoad
```

6.67 /tpm/include/private/prototypes/ContextSave_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ContextSave  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CONTEXTSAVE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CONTEXTSAVE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_CONTEXT saveHandle;
} ContextSave_In;

// Output structure definition
typedef struct
{
    TPMS_CONTEXT context;
} ContextSave_Out;

// Response code modifiers
#    define RC_ContextSave_saveHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ContextSave(ContextSave_In* in, ContextSave_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CONTEXTSAVE_FP_H_
#endif    // CC_ContextSave
```

6.68 /tpm/include/private/prototypes/Context_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _CONTEXT_SPT_FP_H_
#define _CONTEXT_SPT_FP_H_

//*** ComputeContextProtectionKey()
// This function retrieves the symmetric protection key for context encryption
```

```
// It is used by TPM2_ConextSave and TPM2_ContextLoad to create the symmetric
// encryption key and iv
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM
//                              failed to derive the Firmware SVN Secret for the
//                              requested SVN.
TPM_RC ComputeContextProtectionKey(TPMS_CONTEXT*  contextBlob,  // IN: context blob
                                   TPM2B_SYM_KEY* symKey,  // OUT: the symmetric key
                                   TPM2B_IV*      iv       // OUT: the IV.
);

//*** ComputeContextIntegrity()
// Generate the integrity hash for a context
//       It is used by TPM2_ContextSave to create an integrity hash
//       and by TPM2_ContextLoad to compare an integrity hash
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM
//                              failed to derive the Firmware SVN Secret for the
//                              requested SVN.
TPM_RC ComputeContextIntegrity(TPMS_CONTEXT* contextBlob,  // IN: context blob
                               TPM2B_DIGEST* integrity     // OUT: integrity
);

//*** SequenceDataExport()
// This function is used scan through the sequence object and
// either modify the hash state data for export (contextSave) or to
// import it into the internal format (contextLoad).
// This function should only be called after the sequence object has been copied
// to the context buffer (contextSave) or from the context buffer into the sequence
// object. The presumption is that the context buffer version of the data is the
// same size as the internal representation so nothing outsize of the hash context
// area gets modified.
void SequenceDataExport(
    HASH_OBJECT*        object,       // IN: an internal hash object
    HASH_OBJECT_BUFFER* exportObject  // OUT: a sequence context in a buffer
);

//*** SequenceDataImport()
// This function is used scan through the sequence object and
// either modify the hash state data for export (contextSave) or to
// import it into the internal format (contextLoad).
// This function should only be called after the sequence object has been copied
// to the context buffer (contextSave) or from the context buffer into the sequence
// object. The presumption is that the context buffer version of the data is the
// same size as the internal representation so nothing outsize of the hash context
// area gets modified.
void SequenceDataImport(
    HASH_OBJECT*        object,       // IN/OUT: an internal hash object
    HASH_OBJECT_BUFFER* exportObject  // IN/OUT: a sequence context in a buffer
);

#endif  // _CONTEXT_SPT_FP_H_
```

6.69 /tpm/include/private/prototypes/CreateLoaded_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT
```

```
#if CC_CreateLoaded  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATELOADED_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATELOADED_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_PARENT         parentHandle;
    TPM2B_SENSITIVE_CREATE inSensitive;
    TPM2B_TEMPLATE         inPublic;
} CreateLoaded_In;

// Output structure definition
typedef struct
{
    TPM_HANDLE    objectHandle;
    TPM2B_PRIVATE outPrivate;
    TPM2B_PUBLIC  outPublic;
    TPM2B_NAME    name;
} CreateLoaded_Out;

// Response code modifiers
#    define RC_CreateLoaded_parentHandle (TPM_RC_H + TPM_RC_1)
#    define RC_CreateLoaded_inSensitive  (TPM_RC_P + TPM_RC_1)
#    define RC_CreateLoaded_inPublic     (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_CreateLoaded(CreateLoaded_In* in, CreateLoaded_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATELOADED_FP_H_
#endif    // CC_CreateLoaded
```

6.70 /tpm/include/private/prototypes/CreatePrimary_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_CreatePrimary  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATEPRIMARY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATEPRIMARY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_HIERARCHY      primaryHandle;
    TPM2B_SENSITIVE_CREATE inSensitive;
    TPM2B_PUBLIC           inPublic;
    TPM2B_DATA             outsideInfo;
    TPML_PCR_SELECTION     creationPCR;
} CreatePrimary_In;

// Output structure definition
typedef struct
{
    TPM_HANDLE          objectHandle;
    TPM2B_PUBLIC        outPublic;
    TPM2B_CREATION_DATA creationData;
    TPM2B_DIGEST        creationHash;
    TPMT_TK_CREATION    creationTicket;
    TPM2B_NAME          name;
} CreatePrimary_Out;

// Response code modifiers
```

```
#    define RC_CreatePrimary_primaryHandle (TPM_RC_H + TPM_RC_1)
#    define RC_CreatePrimary_inSensitive   (TPM_RC_P + TPM_RC_1)
#    define RC_CreatePrimary_inPublic      (TPM_RC_P + TPM_RC_2)
#    define RC_CreatePrimary_outsideInfo   (TPM_RC_P + TPM_RC_3)
#    define RC_CreatePrimary_creationPCR   (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_CreatePrimary(CreatePrimary_In* in, CreatePrimary_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATEPRIMARY_FP_H_
#endif    // CC_CreatePrimary
```

6.71 /tpm/include/private/prototypes/Create_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Create  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT         parentHandle;
    TPM2B_SENSITIVE_CREATE inSensitive;
    TPM2B_PUBLIC           inPublic;
    TPM2B_DATA             outsideInfo;
    TPML_PCR_SELECTION     creationPCR;
} Create_In;

// Output structure definition
typedef struct
{
    TPM2B_PRIVATE       outPrivate;
    TPM2B_PUBLIC        outPublic;
    TPM2B_CREATION_DATA creationData;
    TPM2B_DIGEST        creationHash;
    TPMT_TK_CREATION    creationTicket;
} Create_Out;

// Response code modifiers
#    define RC_Create_parentHandle  (TPM_RC_H + TPM_RC_1)
#    define RC_Create_inSensitive   (TPM_RC_P + TPM_RC_1)
#    define RC_Create_inPublic      (TPM_RC_P + TPM_RC_2)
#    define RC_Create_outsideInfo   (TPM_RC_P + TPM_RC_3)
#    define RC_Create_creationPCR   (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_Create(Create_In* in, Create_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_CREATE_FP_H_
#endif    // CC_Create
```

6.72 /tpm/include/private/prototypes/CryptCmac_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _CRYPT_CMAC_FP_H_
```

```
#define _CRYPT_CMAC_FP_H_

#if ALG_CMAC

//*** CryptCmacStart()
// This is the function to start the CMAC sequence operation. It initializes the
// dispatch functions for the data and end operations for CMAC and initializes the
// parameters that are used for the processing of data, including the key, key size
// and block cipher algorithm.
UINT16
CryptCmacStart(
    SMAC_STATE* state, TPMU_PUBLIC_PARMS* keyParms, TPM_ALG_ID macAlg, TPM2B* key);

//*** CryptCmacData()
// This function is used to add data to the CMAC sequence computation. The function
// will XOR new data into the IV. If the buffer is full, and there is additional
// input data, the data is encrypted into the IV buffer, the new data is then
// XOR into the IV. When the data runs out, the function returns without encrypting
// even if the buffer is full. The last data block of a sequence will not be
// encrypted until the call to CryptCmacEnd(). This is to allow the proper subkey
// to be computed and applied before the last block is encrypted.
void CryptCmacData(SMAC_STATES* state, UINT32 size, const BYTE* buffer);

//*** CryptCmacEnd()
// This is the completion function for the CMAC. It does padding, if needed, and
// selects the subkey to be applied before the last block is encrypted.
UINT16
CryptCmacEnd(SMAC_STATES* state, UINT32 outSize, BYTE* outBuffer);
#endif

#endif  // _CRYPT_CMAC_FP_H_
```

6.73 /tpm/include/private/prototypes/CryptEccCrypt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Feb 28, 2020  Time: 03:04:48PM
 */

#ifndef _CRYPT_ECC_CRYPT_FP_H_
#define _CRYPT_ECC_CRYPT_FP_H_

#if CC_ECC_Encrypt || CC_ECC_Encrypt

//*** CryptEccSelectScheme()
// This function is used by TPM2_ECC_Decrypt and TPM2_ECC_Encrypt.  It sets scheme
// either the input scheme or the key scheme. If they key scheme is not TPM_ALG_NULL
// then the input scheme must be TPM_ALG_NULL or the same as the key scheme. If
// not, then the function returns FALSE.
//  Return Type: BOOL
//      TRUE        'scheme' is set
//      FALSE       'scheme' is not valid (it may have been changed).
BOOL CryptEccSelectScheme(OBJECT*          key,    //IN: key containing default scheme
                          TPMT_KDF_SCHEME* scheme  // IN: a decrypt scheme
);

//*** CryptEccEncrypt()
//This function performs ECC-based data obfuscation. The only scheme that is currently
// supported is MGF1 based. See Part 1, Annex D for details.
//  Return Type: TPM_RC
//      TPM_RC_CURVE            unsupported curve
//      TPM_RC_HASH             hash not allowed
//      TPM_RC_SCHEME           'scheme' is not supported
//      TPM_RC_NO_RESULT        internal error in big number processing
LIB_EXPORT TPM_RC CryptEccEncrypt(
```

```
    OBJECT*           key,        // IN: public key of recipient
    TPMT_KDF_SCHEME*  scheme,     // IN: scheme to use.
    TPM2B_MAX_BUFFER* plainText,  // IN: the text to obfuscate
    TPMS_ECC_POINT*   c1,         // OUT: public ephemeral key
    TPM2B_MAX_BUFFER* c2,         // OUT: obfuscated text
    TPM2B_DIGEST*     c3          // OUT: digest of ephemeral key
                                  //      and plainText
);

//*** CryptEccDecrypt()
// This function performs ECC decryption and integrity check of the input data.
//  Return Type: TPM_RC
//      TPM_RC_CURVE            unsupported curve
//      TPM_RC_HASH             hash not allowed
//      TPM_RC_SCHEME           'scheme' is not supported
//      TPM_RC_NO_RESULT        internal error in big number processing
//      TPM_RC_VALUE            C3 did not match hash of recovered data
LIB_EXPORT TPM_RC CryptEccDecrypt(
    OBJECT*           key,        // IN: key used for data recovery
    TPMT_KDF_SCHEME*  scheme,     // IN: scheme to use.
    TPM2B_MAX_BUFFER* plainText,  // OUT: the recovered text
    TPMS_ECC_POINT*   c1,         // IN: public ephemeral key
    TPM2B_MAX_BUFFER* c2,         // IN: obfuscated text
    TPM2B_DIGEST*     c3          // IN: digest of ephemeral key
                                  //      and plainText
);
#endif  // CC_ECC_Encrypt || CC_ECC_Encrypt

#endif  // _CRYPT_ECC_CRYPT_FP_H_
```

6.74 /tpm/include/private/prototypes/CryptEccKeyExchange_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _CRYPT_ECC_KEY_EXCHANGE_FP_H_
#define _CRYPT_ECC_KEY_EXCHANGE_FP_H_

#if CC_ZGen_2Phase == YES

//*** CryptEcc2PhaseKeyExchange()
// This function is the dispatch routine for the EC key exchange functions that use
// two ephemeral and two static keys.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME               scheme is not defined
LIB_EXPORT TPM_RC CryptEcc2PhaseKeyExchange(
    TPMS_ECC_POINT*      outZ1,    // OUT: a computed point
    TPMS_ECC_POINT*      outZ2,    // OUT: and optional second point
    TPM_ECC_CURVE        curveId,  // IN: the curve for the computations
    TPM_ALG_ID           scheme,   // IN: the key exchange scheme
    TPM2B_ECC_PARAMETER* dsA,      // IN: static private TPM key
    TPM2B_ECC_PARAMETER* deA,      // IN: ephemeral private TPM key
    TPMS_ECC_POINT*      QsB,      // IN: static public party B key
    TPMS_ECC_POINT*      QeB       // IN: ephemeral public party B key
);
#  if ALG_SM2

//*** SM2KeyExchange()
// This function performs the key exchange defined in SM2.
// The first step is to compute
//  'tA' = ('dsA' + 'deA'  avf(Xe,A)) mod 'n'
// Then, compute the 'Z' value from
// 'outZ' = ('h'  'tA' mod 'n') ('QsA' + [avf('QeB.x')]('QeB')).
```

```
// The function will compute the ephemeral public key from the ephemeral
// private key.
// All points are required to be on the curve of 'inQsA'. The function will fail
// catastrophically if this is not the case
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        the value for dsA does not give a valid point on the
//                              curve
LIB_EXPORT TPM_RC SM2KeyExchange(
    TPMS_ECC_POINT*      outZ,     // OUT: the computed point
    TPM_ECC_CURVE        curveId,  // IN: the curve for the computations
    TPM2B_ECC_PARAMETER* dsAIn,    // IN: static private TPM key
    TPM2B_ECC_PARAMETER* deAIn,    // IN: ephemeral private TPM key
    TPMS_ECC_POINT*      QsBIn,    // IN: static public party B key
    TPMS_ECC_POINT*      QeBIn     // IN: ephemeral public party B key
);
#  endif
#endif  // CC_ZGen_2Phase

#endif  // _CRYPT_ECC_KEY_EXCHANGE_FP_H_
```

6.75 /tpm/include/private/prototypes/CryptEccMain_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 03:18:00PM
 */

#ifndef _CRYPT_ECC_MAIN_FP_H_
#define _CRYPT_ECC_MAIN_FP_H_

#if ALG_ECC

//** Functions
#  if SIMULATION
void EccSimulationEnd(void);
#  endif  // SIMULATION

//*** CryptEccInit()
// This function is called at _TPM_Init
BOOL CryptEccInit(void);

//*** CryptEccStartup()
// This function is called at TPM2_Startup().
BOOL CryptEccStartup(void);

//*** ClearPoint2B(generic)
// Initialize the size values of a TPMS_ECC_POINT structure.
void ClearPoint2B(TPMS_ECC_POINT* p  // IN: the point
);

//*** CryptEccGetParametersByCurveId()
// This function returns a pointer to the curve data that is associated with
// the indicated curveId.
// If there is no curve with the indicated ID, the function returns NULL. This
// function is in this module so that it can be called by GetCurve data.
//  Return Type: const TPM_ECC_CURVE_METADATA
//      NULL            curve with the indicated TPM_ECC_CURVE is not implemented
//      != NULL         pointer to the curve data
LIB_EXPORT const TPM_ECC_CURVE_METADATA* CryptEccGetParametersByCurveId(
    TPM_ECC_CURVE curveId  // IN: the curveID
);

//*** CryptEccGetKeySizeForCurve()
// This function returns the key size in bits of the indicated curve.
LIB_EXPORT UINT16 CryptEccGetKeySizeForCurve(TPM_ECC_CURVE curveId  // IN: the curve
```

```
);

//***CryptEccGetOID()
const BYTE* CryptEccGetOID(TPM_ECC_CURVE curveId);

//*** CryptEccGetCurveByIndex()
// This function returns the number of the 'i'-th implemented curve. The normal
// use would be to call this function with 'i' starting at 0. When the 'i' is greater
// than or equal to the number of implemented curves, TPM_ECC_NONE is returned.
LIB_EXPORT TPM_ECC_CURVE CryptEccGetCurveByIndex(UINT16 i);

//*** CryptCapGetECCCurve()
// This function returns the list of implemented ECC curves.
//  Return Type: TPMI_YES_NO
//      YES             if no more ECC curve is available
//      NO              if there are more ECC curves not reported
TPMI_YES_NO
CryptCapGetECCCurve(TPM_ECC_CURVE   curveID,   // IN: the starting ECC curve
                    UINT32          maxCount,  // IN: count of returned curves
                    TPML_ECC_CURVE* curveList  // OUT: ECC curve list
);

//*** CryptCapGetOneECCCurve()
// This function returns whether the ECC curve is implemented.
BOOL CryptCapGetOneECCCurve(TPM_ECC_CURVE curveID  // IN: the  ECC curve
);

//*** CryptGetCurveSignScheme()
// This function will return a pointer to the scheme of the curve.
const TPMT_ECC_SCHEME* CryptGetCurveSignScheme(
    TPM_ECC_CURVE curveId  // IN: The curve selector
);

//*** CryptGenerateR()
// This function computes the commit random value for a split signing scheme.
//
// If 'c' is NULL, it indicates that 'r' is being generated
// for TPM2_Commit.
// If 'c' is not NULL, the TPM will validate that the 'gr.commitArray'
// bit associated with the input value of 'c' is SET. If not, the TPM
// returns FALSE and no 'r' value is generated.
//  Return Type: BOOL
//      TRUE(1)         r value computed
//      FALSE(0)        no r value computed
BOOL CryptGenerateR(TPM2B_ECC_PARAMETER* r,         // OUT: the generated random value
                    UINT16*              c,         // IN/OUT: count value.
                    TPMI_ECC_CURVE       curveID,   // IN: the curve for the value
                    TPM2B_NAME*          name       // IN: optional name of a key to
                                                    //     associate with 'r'
);

//*** CryptCommit()
// This function is called when the count value is committed. The 'gr.commitArray'
// value associated with the current count value is SET and g_commitCounter is
// incremented. The low-order 16 bits of old value of the counter is returned.
UINT16
CryptCommit(void);

//*** CryptEndCommit()
// This function is called when the signing operation using the committed value
// is completed. It clears the gr.commitArray bit associated with the count
// value so that it can't be used again.
void CryptEndCommit(UINT16 c  // IN: the counter value of the commitment
);

//*** CryptEccGetParameters()
```

```
// This function returns the ECC parameter details of the given curve.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        unsupported ECC curve ID
BOOL CryptEccGetParameters(
    TPM_ECC_CURVE              curveId,    // IN: ECC curve ID
    TPMS_ALGORITHM_DETAIL_ECC* parameters  // OUT: ECC parameters
);

//*** TpmEcc_IsValidPrivateEcc()
// Checks that 0 < 'x' < 'q'
BOOL TpmEcc_IsValidPrivateEcc(const Crypt_Int*      x,  // IN: private key to check
                              const Crypt_EccCurve* E   // IN: the curve to check
);

LIB_EXPORT BOOL CryptEccIsValidPrivateKey(TPM2B_ECC_PARAMETER* d,
                                          TPM_ECC_CURVE        curveId);

//*** TpmEcc_PointMult()
// This function does a point multiply of the form 'R' = ['d']'S' + ['u']'Q' where the
// parameters are Crypt_Int* values. If 'S' is NULL and d is not NULL, then it
computes
// 'R' = ['d']'G' + ['u']'Q'  or just 'R' = ['d']'G' if 'u' and 'Q' are NULL.
// If 'skipChecks' is TRUE, then the function will not verify that the inputs are
// correct for the domain. This would be the case when the values were created by the
// CryptoEngine code.
// It will return TPM_RC_NO_RESULT if the resulting point is the point at infinity.
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        result of multiplication is a point at infinity
//      TPM_RC_ECC_POINT        'S' or 'Q' is not on the curve
//      TPM_RC_VALUE            'd' or 'u' is not < n
TPM_RC
TpmEcc_PointMult(Crypt_Point*          R,  // OUT: computed point
                 const Crypt_Point*    S,  // IN: optional point to multiply by 'd'
                 const Crypt_Int*      d,  // IN: scalar for [d]S or [d]G
                 const Crypt_Point*    Q,  // IN: optional second point
                 const Crypt_Int*      u,  // IN: optional second scalar
                 const Crypt_EccCurve* E   // IN: curve parameters
);

//***TpmEcc_GenPrivateScalar()
// This function gets random values that are the size of the key plus 64 bits. The
// value is reduced (mod ('q' - 1)) and incremented by 1 ('q' is the order of the
// curve. This produces a value ('d') such that 1 <= 'd' < 'q'. This is the method
// of FIPS 186-4 Section B.4.1 ""Key Pair Generation Using Extra Random Bits"".
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure generating private key
BOOL TpmEcc_GenPrivateScalar(
    Crypt_Int*            dOut,  // OUT: the qualified random value
    const Crypt_EccCurve* E,     // IN: curve for which the private key
                                 //     needs to be appropriate
    RAND_STATE* rand             // IN: state for DRBG
);

//*** TpmEcc_GenerateKeyPair()
// This function gets a private scalar from the source of random bits and does
// the point multiply to get the public key.
BOOL TpmEcc_GenerateKeyPair(Crypt_Int*           bnD,  // OUT: private scalar
                            Crypt_Point*         ecQ,  // OUT: public point
                            const Crypt_EccCurve* E,   // IN: curve for the point
                            RAND_STATE*          rand  // IN: DRBG state to use
);

//***CryptEccNewKeyPair(***)
// This function creates an ephemeral ECC. It is ephemeral in that
```

```
// is expected that the private part of the key will be discarded
LIB_EXPORT TPM_RC CryptEccNewKeyPair(
    TPMS_ECC_POINT*      Qout,    // OUT: the public point
    TPM2B_ECC_PARAMETER* dOut,    // OUT: the private scalar
    TPM_ECC_CURVE        curveId  // IN: the curve for the key
);

//*** CryptEccPointMultiply()
// This function computes 'R' := ['dIn']'G' + ['uIn']'QIn'. Where 'dIn' and
// 'uIn' are scalars, 'G' and 'QIn' are points on the specified curve and 'G' is the
// default generator of the curve.
//
// The 'xOut' and 'yOut' parameters are optional and may be set to NULL if not
// used.
//
// It is not necessary to provide 'uIn' if 'QIn' is specified but one of 'uIn' and
// 'dIn' must be provided. If 'dIn' and 'QIn' are specified but 'uIn' is not
// provided, then 'R' = ['dIn']'QIn'.
//
// If the multiply produces the point at infinity, the TPM_RC_NO_RESULT is returned.
//
// The sizes of 'xOut' and yOut' will be set to be the size of the degree of
// the curve
//
// It is a fatal error if 'dIn' and 'uIn' are both unspecified (NULL) or if 'Qin'
// or 'Rout' is unspecified.
//
//  Return Type: TPM_RC
//      TPM_RC_ECC_POINT         the point 'Pin' or 'Qin' is not on the curve
//      TPM_RC_NO_RESULT         the product point is at infinity
//      TPM_RC_CURVE             bad curve
//      TPM_RC_VALUE             'dIn' or 'uIn' out of range
//
LIB_EXPORT TPM_RC CryptEccPointMultiply(
    TPMS_ECC_POINT*      Rout,     // OUT: the product point R
    TPM_ECC_CURVE        curveId,  // IN: the curve to use
    TPMS_ECC_POINT*      Pin,      // IN: first point (can be null)
    TPM2B_ECC_PARAMETER* dIn,      // IN: scalar value for [dIn]Qin
                                   //     the Pin
    TPMS_ECC_POINT*      Qin,      // IN: point Q
    TPM2B_ECC_PARAMETER* uIn       // IN: scalar value for the multiplier
                                   //     of Q
);

//*** CryptEccIsPointOnCurve()
// This function is used to test if a point is on a defined curve. It does this
// by checking that 'y'^2 mod 'p' = 'x'^3 + 'a'*'x' + 'b' mod 'p'.
//
// It is a fatal error if 'Q' is not specified (is NULL).
//  Return Type: BOOL
//      TRUE(1)          point is on curve
//      FALSE(0)         point is not on curve or curve is not supported
LIB_EXPORT BOOL CryptEccIsPointOnCurve(
    TPM_ECC_CURVE   curveId,  // IN: the curve selector
    TPMS_ECC_POINT* Qin       // IN: the point.
);

//*** CryptEccGenerateKey()
// This function generates an ECC key pair based on the input parameters.
// This routine uses KDFa to produce candidate numbers. The method is according
// to FIPS 186-3, section B.1.2 "Key Pair Generation by Testing Candidates."
// According to the method in FIPS 186-3, the resulting private value 'd' should be
// 1 <= 'd' < 'n' where 'n' is the order of the base point.
//
// It is a fatal error if 'Qout', 'dOut', is not provided (is NULL).
//
```

```
// If the curve is not supported
// If 'seed' is not provided, then a random number will be used for the key
//  Return Type: TPM_RC
//      TPM_RC_CURVE            curve is not supported
//      TPM_RC_NO_RESULT        could not verify key with signature (FIPS only)
LIB_EXPORT TPM_RC CryptEccGenerateKey(
    TPMT_PUBLIC* publicArea,    // IN/OUT: The public area template for
                                //      the new key. The public key
                                //      area will be replaced computed
                                //      ECC public key
    TPMT_SENSITIVE* sensitive,  // OUT: the sensitive area will be
                                //      updated to contain the private
                                //      ECC key and the symmetric
                                //      encryption key
    RAND_STATE* rand            // IN: if not NULL, the deterministic
                                //     RNG state
);
#endif  // ALG_ECC

#endif  // _CRYPT_ECC_MAIN_FP_H_
```

6.76 /tpm/include/private/prototypes/CryptEccSignature_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _CRYPT_ECC_SIGNATURE_FP_H_
#define _CRYPT_ECC_SIGNATURE_FP_H_

#if ALG_ECC

//*** CryptEccSign()
// This function is the dispatch function for the various ECC-based
// signing schemes.
// There is a bit of ugliness to the parameter passing. In order to test this,
// we sometime would like to use a deterministic RNG so that we can get the same
// signatures during testing. The easiest way to do this for most schemes is to
// pass in a deterministic RNG and let it return canned values during testing.
// There is a competing need for a canned parameter to use in ECDAA. To accommodate
// both needs with minimal fuss, a special type of RAND_STATE is defined to carry
// the address of the commit value. The setup and handling of this is not very
// different for the caller than what was in previous versions of the code.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME           'scheme' is not supported
LIB_EXPORT TPM_RC CryptEccSign(TPMT_SIGNATURE* signature,  // OUT: signature
                               OBJECT* signKey,  // IN: ECC key to sign the hash
                               const TPM2B_DIGEST* digest,  // IN: digest to sign
                               TPMT_ECC_SCHEME*    scheme,  // IN: signing scheme
                               RAND_STATE*         rand);

//*** CryptEccValidateSignature()
// This function validates an EcDsa or EcSchnorr signature.
// The point 'Qin' needs to have been validated to be on the curve of 'curveId'.
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE            not a valid signature
LIB_EXPORT TPM_RC CryptEccValidateSignature(
    TPMT_SIGNATURE*     signature,  // IN: signature to be verified
    OBJECT*             signKey,    // IN: ECC key signed the hash
    const TPM2B_DIGEST* digest      // IN: digest that was signed
);

//***CryptEccCommitCompute()
// This function performs the point multiply operations required by TPM2_Commit.
```

```
//
// If 'B' or 'M' is provided, they must be on the curve defined by 'curveId'. This
// routine does not check that they are on the curve and results are unpredictable
// if they are not.
//
// It is a fatal error if 'r' is NULL. If 'B' is not NULL, then it is a
// fatal error if 'd' is NULL or if 'K' and 'L' are both NULL.
// If 'M' is not NULL, then it is a fatal error if 'E' is NULL.
//
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        if 'K', 'L' or 'E' was computed to be the point
//                              at infinity
//      TPM_RC_CANCELED         a cancel indication was asserted during this
//                              function
LIB_EXPORT TPM_RC CryptEccCommitCompute(
    TPMS_ECC_POINT*      K,         // OUT: [d]B or [r]Q
    TPMS_ECC_POINT*      L,         // OUT: [r]B
    TPMS_ECC_POINT*      E,         // OUT: [r]M
    TPM_ECC_CURVE        curveId,   // IN: the curve for the computations
    TPMS_ECC_POINT*      M,         // IN: M (optional)
    TPMS_ECC_POINT*      B,         // IN: B (optional)
    TPM2B_ECC_PARAMETER* d,         // IN: d (optional)
    TPM2B_ECC_PARAMETER* r          // IN: the computed r value (required)
);
#endif  // ALG_ECC

#endif  // _CRYPT_ECC_SIGNATURE_FP_H_
```

6.77 /tpm/include/private/prototypes/CryptHash_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Feb 28, 2020  Time: 03:04:48PM
 */

#ifndef _CRYPT_HASH_FP_H_
#define _CRYPT_HASH_FP_H_

//*** CryptHashInit()
// This function is called by _TPM_Init do perform the initialization operations for
// the library.
BOOL CryptHashInit(void);

//*** CryptHashStartup()
// This function is called by TPM2_Startup(). It checks that the size of the
// HashDefArray is consistent with the HASH_COUNT.
BOOL CryptHashStartup(void);

//*** CryptGetHashDef()
// This function accesses the hash descriptor associated with a hash a
// algorithm. The function returns a pointer to a 'null' descriptor if hashAlg is
// TPM_ALG_NULL or not a defined algorithm.
PHASH_DEF
CryptGetHashDef(TPM_ALG_ID hashAlg);

//*** CryptHashIsValidAlg()
// This function tests to see if an algorithm ID is a valid hash algorithm. If
// flag is true, then TPM_ALG_NULL is a valid hash.
//  Return Type: BOOL
//      TRUE(1)         hashAlg is a valid, implemented hash on this TPM
//      FALSE(0)        hashAlg is not valid for this TPM
BOOL CryptHashIsValidAlg(TPM_ALG_ID hashAlg,  // IN: the algorithm to check
                         BOOL       flag  // IN: TRUE if TPM_ALG_NULL is to be treated
                                          //     as a valid hash
);
```

```
//*** CryptHashGetAlgByIndex()
// This function is used to iterate through the hashes. TPM_ALG_NULL
// is returned for all indexes that are not valid hashes.
// If the TPM implements 3 hashes, then an 'index' value of 0 will
// return the first implemented hash and an 'index' of 2 will return the
// last. All other index values will return TPM_ALG_NULL.
//
//  Return Type: TPM_ALG_ID
// TPM_ALG_xxx          a hash algorithm
// TPM_ALG_NULL         this can be used as a stop value
LIB_EXPORT TPM_ALG_ID CryptHashGetAlgByIndex(UINT32 index  // IN: the index
);

//*** CryptHashGetDigestSize()
// Returns the size of the digest produced by the hash. If 'hashAlg' is not a hash
// algorithm, the TPM will FAIL.
//  Return Type: UINT16
//   0       TPM_ALG_NULL
//   > 0     the digest size
//
LIB_EXPORT UINT16 CryptHashGetDigestSize(
    TPM_ALG_ID hashAlg  // IN: hash algorithm to look up
);

//*** CryptHashGetBlockSize()
// Returns the size of the block used by the hash. If 'hashAlg' is not a hash
// algorithm, the TPM will FAIL.
//  Return Type: UINT16
//   0       TPM_ALG_NULL
//   > 0     the digest size
//
LIB_EXPORT UINT16 CryptHashGetBlockSize(
    TPM_ALG_ID hashAlg  // IN: hash algorithm to look up
);

//*** CryptHashGetOid()
// This function returns a pointer to DER=encoded OID for a hash algorithm. All OIDs
// are full OID values including the Tag (0x06) and length byte.
LIB_EXPORT const BYTE* CryptHashGetOid(TPM_ALG_ID hashAlg);

//***  CryptHashGetContextAlg()
// This function returns the hash algorithm associated with a hash context.
TPM_ALG_ID
CryptHashGetContextAlg(PHASH_STATE state  // IN: the context to check
);

//*** CryptHashCopyState
// This function is used to clone a HASH_STATE.
LIB_EXPORT void CryptHashCopyState(HASH_STATE* out,  // OUT: destination of the state
                                   const HASH_STATE* in  // IN: source of the state
);

//*** CryptHashExportState()
// This function is used to export a hash or HMAC hash state. This function
// would be called when preparing to context save a sequence object.
void CryptHashExportState(
    PCHASH_STATE internalFmt,       // IN: the hash state formatted for use by
                                    //     library
    PEXPORT_HASH_STATE externalFmt  // OUT: the exported hash state
);

//*** CryptHashImportState()
// This function is used to import the hash state. This function
// would be called to import a hash state when the context of a sequence object
// was being loaded.
```

```
void CryptHashImportState(
    PHASH_STATE internalFmt,         // OUT: the hash state formatted for use by
                                     //     the library
    PCEXPORT_HASH_STATE externalFmt  // IN: the exported hash state
);

//*** CryptHashStart()
// Functions starts a hash stack
// Start a hash stack and returns the digest size. As a side effect, the
// value of 'stateSize' in hashState is updated to indicate the number of bytes
// of state that were saved. This function calls GetHashServer() and that function
// will put the TPM into failure mode if the hash algorithm is not supported.
//
// This function does not use the sequence parameter. If it is necessary to import
// or export context, this will start the sequence in a local state
// and export the state to the input buffer. Will need to add a flag to the state
// structure to indicate that it needs to be imported before it can be used.
// (BLEH).
//  Return Type: UINT16
//  0           hash is TPM_ALG_NULL
// >0           digest size
LIB_EXPORT UINT16 CryptHashStart(
    PHASH_STATE hashState,  // OUT: the running hash state
    TPM_ALG_ID  hashAlg     // IN: hash algorithm
);

//*** CryptDigestUpdate()
// Add data to a hash or HMAC, SMAC stack.
//
void CryptDigestUpdate(PHASH_STATE hashState,  // IN: the hash context information
                       UINT32      dataSize,   // IN: the size of data to be added
                       const BYTE* data        // IN: data to be hashed
);

//*** CryptHashEnd()
// Complete a hash or HMAC computation. This function will place the smaller of
// 'digestSize' or the size of the digest in 'dOut'. The number of bytes in the
// placed in the buffer is returned. If there is a failure, the returned value
// is <= 0.
//  Return Type: UINT16
//       0      no data returned
//     > 0      the number of bytes in the digest or dOutSize, whichever is smaller
LIB_EXPORT UINT16 CryptHashEnd(PHASH_STATE hashState,  // IN: the state of hash stack
                               UINT32      dOutSize,   // IN: size of digest buffer
                               BYTE*       dOut        // OUT: hash digest
);

//*** CryptHashBlock()
// Start a hash, hash a single block, update 'digest' and return the size of
// the results.
//
// The 'digestSize' parameter can be smaller than the digest. If so, only the more
// significant bytes are returned.
//  Return Type: UINT16
//  >= 0        number of bytes placed in 'dOut'
LIB_EXPORT UINT16 CryptHashBlock(TPM_ALG_ID  hashAlg,   // IN: The hash algorithm
                                 UINT32      dataSize,  // IN: size of buffer to hash
                                 const BYTE* data,      // IN: the buffer to hash
                                 UINT32 dOutSize,  // IN: size of the digest buffer
                                 BYTE*  dOut       // OUT: digest buffer
);

//*** CryptDigestUpdate2B()
// This function updates a digest (hash or HMAC) with a TPM2B.
//
// This function can be used for both HMAC and hash functions so the
```

```
// 'digestState' is void so that either state type can be passed.
LIB_EXPORT void CryptDigestUpdate2B(PHASH_STATE  state,  // IN: the digest state
                                    const TPM2B* bIn     // IN: 2B containing the data
);

//*** CryptHashEnd2B()
// This function is the same as CryptCompleteHash() but the digest is
// placed in a TPM2B. This is the most common use and this is provided
// for specification clarity. 'digest.size' should be set to indicate the number of
// bytes to place in the buffer
//  Return Type: UINT16
//      >=0     the number of bytes placed in 'digest.buffer'
LIB_EXPORT UINT16 CryptHashEnd2B(
    PHASH_STATE state,  // IN: the hash state
    P2B         digest  // IN: the size of the buffer Out: requested
                        //     number of bytes
);

//*** CryptDigestUpdateInt()
// This function is used to include an integer value to a hash stack. The function
// marshals the integer into its canonical form before calling CryptDigestUpdate().
LIB_EXPORT void CryptDigestUpdateInt(
    void*  state,    // IN: the state of hash stack
    UINT32 intSize,  // IN: the size of 'intValue' in bytes
    UINT64 intValue  // IN: integer value to be hashed
);

//*** CryptHmacStart()
// This function is used to start an HMAC using a temp
// hash context. The function does the initialization
// of the hash with the HMAC key XOR iPad and updates the
// HMAC key XOR oPad.
//
// The function returns the number of bytes in a digest produced by 'hashAlg'.
//  Return Type: UINT16
//  >= 0        number of bytes in digest produced by 'hashAlg' (may be zero)
//
LIB_EXPORT UINT16 CryptHmacStart(PHMAC_STATE state,    // IN/OUT: the state buffer
                                 TPM_ALG_ID  hashAlg,  // IN: the algorithm to use
                                 UINT16      keySize,  // IN: the size of the HMAC key
                                 const BYTE* key       // IN: the HMAC key
);

//*** CryptHmacEnd()
// This function is called to complete an HMAC. It will finish the current
// digest, and start a new digest. It will then add the oPadKey and the
// completed digest and return the results in dOut. It will not return more
// than dOutSize bytes.
//  Return Type: UINT16
//  >= 0        number of bytes in 'dOut' (may be zero)
LIB_EXPORT UINT16 CryptHmacEnd(PHMAC_STATE state,     // IN: the hash state buffer
                               UINT32      dOutSize,  // IN: size of digest buffer
                               BYTE*       dOut       // OUT: hash digest
);

//*** CryptHmacStart2B()
// This function starts an HMAC and returns the size of the digest
// that will be produced.
//
// This function is provided to support the most common use of starting an HMAC
// with a TPM2B key.
//
// The caller must provide a block of memory in which the hash sequence state
// is kept.  The caller should not alter the contents of this buffer until the
// hash sequence is completed or abandoned.
//
```

```
//  Return Type: UINT16
//      > 0     the digest size of the algorithm
//      = 0     the hashAlg was TPM_ALG_NULL
LIB_EXPORT UINT16 CryptHmacStart2B(
    PHMAC_STATE hmacState,  // OUT: the state of HMAC stack. It will be used
                            //     in HMAC update and completion
    TPMI_ALG_HASH hashAlg,  // IN: hash algorithm
    P2B           key       // IN: HMAC key
);

//*** CryptHmacEnd2B()
//   This function is the same as CryptHmacEnd() but the HMAC result
//   is returned in a TPM2B which is the most common use.
//  Return Type: UINT16
//      >=0     the number of bytes placed in 'digest'
LIB_EXPORT UINT16 CryptHmacEnd2B(
    PHMAC_STATE hmacState,  // IN: the state of HMAC stack
    P2B         digest      // OUT: HMAC
);

//** Mask and Key Generation Functions
//*** CryptMGF_KDF()
// This function performs MGF1/KDF1 or KDF2 using the selected hash. KDF1 and KDF2 are
// T('n') = T('n'-1) || H('seed' || 'counter') with the difference being that, with
// KDF1, 'counter' starts at 0 but with KDF2, 'counter' starts at 1. The caller
// determines which version by setting the initial value of counter to either 0 or 1.
// Note: Any value that is not 0 is considered to be 1.
//
// This function returns the length of the mask produced which
// could be zero if the digest algorithm is not supported
//  Return Type: UINT16
//      0       hash algorithm was TPM_ALG_NULL
//    > 0       should be the same as 'mSize'
LIB_EXPORT UINT16 CryptMGF_KDF(UINT32 mSize,  // IN: length of the mask to be produced
                               BYTE*  mask,   // OUT: buffer to receive the mask
                               TPM_ALG_ID hashAlg,   // IN: hash to use
                               UINT32     seedSize,  // IN: size of the seed
                               BYTE*      seed,      // IN: seed size
                               UINT32     counter    // IN: counter initial value
);

//*** CryptKDFa()
// This function performs the key generation according to Part 1 of the
// TPM specification.
//
// This function returns the number of bytes generated which may be zero.
//
// The 'key' and 'keyStream' pointers are not allowed to be NULL. The other
// pointer values may be NULL. The value of 'sizeInBits' must be no larger
// than (2^18)-1 = 256K bits (32385 bytes).
//
// The 'once' parameter is set to allow incremental generation of a large
// value. If this flag is TRUE, 'sizeInBits' will be used in the HMAC computation
// but only one iteration of the KDF is performed. This would be used for
// XOR obfuscation so that the mask value can be generated in digest-sized
// chunks rather than having to be generated all at once in an arbitrarily
// large buffer and then XORed into the result. If 'once' is TRUE, then
// 'sizeInBits' must be a multiple of 8.
//
// Any error in the processing of this command is considered fatal.
//  Return Type: UINT16
//      0           hash algorithm is not supported or is TPM_ALG_NULL
//    > 0           the number of bytes in the 'keyStream' buffer
LIB_EXPORT UINT16 CryptKDFa(
    TPM_ALG_ID   hashAlg,       // IN: hash algorithm used in HMAC
    const TPM2B* key,           // IN: HMAC key
```

```
    const TPM2B* label,         // IN: a label for the KDF
    const TPM2B* contextU,      // IN: context U
    const TPM2B* contextV,      // IN: context V
    UINT32       sizeInBits,    // IN: size of generated key in bits
    BYTE*        keyStream,     // OUT: key buffer
    UINT32*      counterInOut,  // IN/OUT: caller may provide the iteration
                                //     counter for incremental operations to
                                //     avoid large intermediate buffers.
    UINT16 blocks               // IN: If non-zero, this is the maximum number
                                //     of blocks to be returned, regardless
                                //     of sizeInBits
);

//*** CryptKDFe()
// This function implements KDFe() as defined in TPM specification part 1.
//
// This function returns the number of bytes generated which may be zero.
//
// The 'Z' and 'keyStream' pointers are not allowed to be NULL. The other
// pointer values may be NULL. The value of 'sizeInBits' must be no larger
// than (2^18)-1 = 256K bits (32385 bytes).
// Any error in the processing of this command is considered fatal.
//  Return Type: UINT16
//     0            hash algorithm is not supported or is TPM_ALG_NULL
//    > 0           the number of bytes in the 'keyStream' buffer
//
LIB_EXPORT UINT16 CryptKDFe(TPM_ALG_ID   hashAlg,  // IN: hash algorithm used in HMAC
                            TPM2B*       Z,        // IN: Z
                            const TPM2B* label,    // IN: a label value for the KDF
                            TPM2B*       partyUInfo,  // IN: PartyUInfo
                            TPM2B*       partyVInfo,  // IN: PartyVInfo
                            UINT32 sizeInBits,  // IN: size of generated key in bits
                            BYTE*  keyStream    // OUT: key buffer
);

#endif  // _CRYPT_HASH_FP_H_
```

6.78 /tpm/include/private/prototypes/CryptPrimeSieve_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Aug 30, 2019  Time: 02:11:54PM
 */

#ifndef _CRYPT_PRIME_SIEVE_FP_H_
#define _CRYPT_PRIME_SIEVE_FP_H_

#if RSA_KEY_SIEVE

//*** RsaAdjustPrimeLimit()
// This used during the sieve process. The iterator for getting the
// next prime (RsaNextPrime()) will return primes until it hits the
// limit (primeLimit) set up by this function. This causes the sieve
// process to stop when an appropriate number of primes have been
// sieved.
LIB_EXPORT void RsaAdjustPrimeLimit(uint32_t requestedPrimes);

//*** RsaNextPrime()
// This the iterator used during the sieve process. The input is the
// last prime returned (or any starting point) and the output is the
// next higher prime. The function returns 0 when the primeLimit is
// reached.
LIB_EXPORT uint32_t RsaNextPrime(uint32_t lastPrime);

//*** FindNthSetBit()
```

```
// This function finds the nth SET bit in a bit array. The 'n' parameter is
// between 1 and the number of bits in the array (always a multiple of 8).
// If called when the array does not have n bits set, it will return -1
//  Return Type: unsigned int
//      <0      no bit is set or no bit with the requested number is set
//      >=0     the number of the bit in the array that is the nth set
LIB_EXPORT int FindNthSetBit(
    const UINT16 aSize,  // IN: the size of the array to check
    const BYTE*  a,      // IN: the array to check
    const UINT32 n       // IN, the number of the SET bit
);

//*** PrimeSieve()
// This function does a prime sieve over the input 'field' which has as its
// starting address the value in bnN. Since this initializes the Sieve
// using a precomputed field with the bits associated with 3, 5 and 7 already
// turned off, the value of pnN may need to be adjusted by a few counts to allow
// the precomputed field to be used without modification.
//
// To get better performance, one could address the issue of developing the
// composite numbers. When the size of the prime gets large, the time for doing
// the divisions goes up, noticeably. It could be better to develop larger composite
// numbers even if they need to be Crypt_Int*'s themselves. The object would be to
// reduce the number of times that the large prime is divided into a few large
// divides and then use smaller divides to get to the final 16 bit (or smaller)
// remainders.
LIB_EXPORT UINT32 PrimeSieve(Crypt_Int* bnN,    // IN/OUT: number to sieve
                             UINT32 fieldSize,  // IN: size of the field area in bytes
                             BYTE*  field       // IN: field
);
#  ifdef SIEVE_DEBUG

//***SetFieldSize()
// Function to set the field size used for prime generation. Used for tuning.
LIB_EXPORT uint32_t SetFieldSize(uint32_t newFieldSize);
#  endif  // SIEVE_DEBUG

//*** PrimeSelectWithSieve()
// This function will sieve the field around the input prime candidate. If the
// sieve field is not empty, one of the one bits in the field is chosen for testing
// with Miller-Rabin. If the value is prime, 'pnP' is updated with this value
// and the function returns success. If this value is not prime, another
// pseudo-random candidate is chosen and tested. This process repeats until
// all values in the field have been checked. If all bits in the field have
// been checked and none is prime, the function returns FALSE and a new random
// value needs to be chosen.
//  Return Type: TPM_RC
//      TPM_RC_FAILURE      TPM in failure mode, probably due to entropy source
//      TPM_RC_SUCCESS      candidate is probably prime
//      TPM_RC_NO_RESULT    candidate is not prime and couldn't find and alternative
//                          in the field
LIB_EXPORT TPM_RC PrimeSelectWithSieve(
    Crypt_Int*  candidate,  // IN/OUT: The candidate to filter
    UINT32      e,          // IN: the exponent
    RAND_STATE* rand        // IN: the random number generator state
);
#  if RSA_INSTRUMENT

//*** PrintTuple()
char* PrintTuple(UINT32* i);

//*** RsaSimulationEnd()
void RsaSimulationEnd(void);

//*** GetSieveStats()
LIB_EXPORT void GetSieveStats(
```

```
    uint32_t* trials, uint32_t* emptyFields, uint32_t* averageBits);
#  endif
#endif  // RSA_KEY_SIEVE
#if !RSA_INSTRUMENT

//*** RsaSimulationEnd()
// Stub for call when not doing instrumentation.
void RsaSimulationEnd(void);
#endif

#endif  // _CRYPT_PRIME_SIEVE_FP_H_
```

6.79 /tpm/include/private/prototypes/CryptPrime_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Aug 30, 2019  Time: 02:11:54PM
 */

#ifndef _CRYPT_PRIME_FP_H_
#define _CRYPT_PRIME_FP_H_

//*** IsPrimeInt()
// This will do a test of a word of up to 32-bits in size.
BOOL IsPrimeInt(uint32_t n);

//*** TpmMath_IsProbablyPrime()
// This function is used when the key sieve is not implemented. This function
// Will try to eliminate some of the obvious things before going on
// to perform MillerRabin as a final verification of primeness.
BOOL TpmMath_IsProbablyPrime(Crypt_Int*  prime,  // IN:
                             RAND_STATE* rand    // IN: the random state just
                             //     in case Miller-Rabin is required
);

//*** MillerRabinRounds()
// Function returns the number of Miller-Rabin rounds necessary to give an
// error probability equal to the security strength of the prime. These values
// are from FIPS 186-3.
UINT32
MillerRabinRounds(UINT32 bits  // IN: Number of bits in the RSA prime
);

//*** MillerRabin()
// This function performs a Miller-Rabin test from FIPS 186-3. It does
// 'iterations' trials on the number. In all likelihood, if the number
// is not prime, the first test fails.
//  Return Type: BOOL
//      TRUE(1)         probably prime
//      FALSE(0)        composite
BOOL MillerRabin(Crypt_Int* bnW, RAND_STATE* rand);
#if ALG_RSA

//*** RsaCheckPrime()
// This will check to see if a number is prime and appropriate for an
// RSA prime.
//
// This has different functionality based on whether we are using key
// sieving or not. If not, the number checked to see if it is divisible by
// the public exponent, then the number is adjusted either up or down
// in order to make it a better candidate. It is then checked for being
// probably prime.
//
// If sieving is used, the number is used to root a sieving process.
//
```

```
TPM_RC
RsaCheckPrime(Crypt_Int* prime, UINT32 exponent, RAND_STATE* rand);

//*** TpmRsa_GeneratePrimeForRSA()
// Function to generate a prime of the desired size with the proper attributes
// for an RSA prime.
TPM_RC
TpmRsa_GeneratePrimeForRSA(
    Crypt_Int* prime,      // IN/OUT: points to the BN that will get the
                           //  random value
    UINT32      bits,      // IN: number of bits to get
    UINT32      exponent,  // IN: the exponent
    RAND_STATE* rand       // IN: the random state
);
#endif  // ALG_RSA

#endif  // _CRYPT_PRIME_FP_H_
```

6.80 /tpm/include/private/prototypes/CryptRand_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _CRYPT_RAND_FP_H_
#define _CRYPT_RAND_FP_H_

//*** DRBG_GetEntropy()
// Even though this implementation never fails, it may get blocked
// indefinitely long in the call to get entropy from the platform
// (DRBG_GetEntropy32()).
// This function is only used during instantiation of the DRBG for
// manufacturing and on each start-up after an non-orderly shutdown.
//
//  Return Type: BOOL
//      TRUE(1)         requested entropy returned
//      FALSE(0)        entropy Failure
BOOL DRBG_GetEntropy(UINT32 requiredEntropy,  // IN: requested number of bytes of full
                                              //     entropy
                     BYTE* entropy  // OUT: buffer to return collected entropy
);

//*** IncrementIv()
// This function increments the IV value by 1. It is used by EncryptDRBG().
void IncrementIv(DRBG_IV* iv);

//*** DRBG_Reseed()
// This function is used when reseeding of the DRBG is required. If
// entropy is provided, it is used in lieu of using hardware entropy.
// Note: the provided entropy must be the required size.
//
//  Return Type: BOOL
//      TRUE(1)         reseed succeeded
//      FALSE(0)        reseed failed, probably due to the entropy generation
BOOL DRBG_Reseed(DRBG_STATE* drbgState,        // IN: the state to update
                 DRBG_SEED*  providedEntropy,  // IN: entropy
                 DRBG_SEED*  additionalData    // IN:
);

//*** DRBG_SelfTest()
// This is run when the DRBG is instantiated and at startup.
//
//  Return Type: BOOL
//      TRUE(1)         test OK
```

```
//      FALSE(0)        test failed
BOOL DRBG_SelfTest(void);

//*** CryptRandomStir()
// This function is used to cause a reseed. A DRBG_SEED amount of entropy is
// collected from the hardware and then additional data is added.
//
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        failure of the entropy generator
LIB_EXPORT TPM_RC CryptRandomStir(UINT16 additionalDataSize, BYTE* additionalData);

//*** CryptRandomGenerate()
// Generate a 'randomSize' number or random bytes.
LIB_EXPORT UINT16 CryptRandomGenerate(UINT16 randomSize, BYTE* buffer);

//*** DRBG_InstantiateSeededKdf()
// This function is used to instantiate a KDF-based RNG. This is used for derivations.
// This function always returns TRUE.
LIB_EXPORT BOOL DRBG_InstantiateSeededKdf(
    KDF_STATE*    state,    // OUT: buffer to hold the state
    TPM_ALG_ID    hashAlg,  // IN: hash algorithm
    TPM_ALG_ID    kdf,      // IN: the KDF to use
    TPM2B*        seed,     // IN: the seed to use
    const TPM2B* label,     // IN: a label for the generation process.
    TPM2B*        context,  // IN: the context value
    UINT32        limit     // IN: Maximum number of bits from the KDF
);

//*** DRBG_AdditionalData()
// Function to reseed the DRBG with additional entropy. This is normally called
// before computing the protection value of a primary key in the Endorsement
// hierarchy.
LIB_EXPORT void DRBG_AdditionalData(DRBG_STATE* drbgState,  // IN:OUT state to update
                                    TPM2B* additionalData  // IN: value to incorporate
);

//*** DRBG_InstantiateSeeded()
// This function is used to instantiate a random number generator from seed values.
// The nominal use of this generator is to create sequences of pseudo-random
// numbers from a seed value.
//
// Return Type: TPM_RC
//  TPM_RC_FAILURE      DRBG self-test failure
LIB_EXPORT TPM_RC DRBG_InstantiateSeeded(
    DRBG_STATE*  drbgState,  // IN/OUT: buffer to hold the state
    const TPM2B* seed,       // IN: the seed to use
    const TPM2B* purpose,    // IN: a label for the generation process.
    const TPM2B* name,       // IN: name of the object
    const TPM2B* additional  // IN: additional data
);

//*** CryptRandStartup()
// This function is called when TPM_Startup is executed. This function always returns
// TRUE.
LIB_EXPORT BOOL CryptRandStartup(void);

//**** CryptRandInit()
// This function is called when _TPM_Init is being processed.
//
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
LIB_EXPORT BOOL CryptRandInit(void);

//*** DRBG_Generate()
// This function generates a random sequence according SP800-90A.
```

```
// If 'random' is not NULL, then 'randomSize' bytes of random values are generated.
// If 'random' is NULL or 'randomSize' is zero, then the function returns
// zero without generating any bits or updating the reseed counter.
// This function returns the number of bytes produced which could be less than the
// number requested if the request is too large ("too large" is implementation
// dependent.)
LIB_EXPORT UINT16 DRBG_Generate(
    RAND_STATE* state,
    BYTE*       random,     // OUT: buffer to receive the random values
    UINT16      randomSize  // IN: the number of bytes to generate
);

//*** DRBG_Instantiate()
// This is CTR_DRBG_Instantiate_algorithm() from [SP 800-90A 10.2.1.3.1].
// This is called when a the TPM DRBG is to be instantiated. This is
// called to instantiate a DRBG used by the TPM for normal
// operations.
//
//  Return Type: BOOL
//      TRUE(1)         instantiation succeeded
//      FALSE(0)        instantiation failed
LIB_EXPORT BOOL DRBG_Instantiate(
    DRBG_STATE* drbgState,       // OUT: the instantiated value
    UINT16      pSize,           // IN: Size of personalization string
    BYTE*       personalization  // IN: The personalization string
);

//*** DRBG_Uninstantiate()
// This is Uninstantiate_function() from [SP 800-90A 9.4].
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE        not a valid state
LIB_EXPORT TPM_RC DRBG_Uninstantiate(
    DRBG_STATE* drbgState  // IN/OUT: working state to erase
);

#endif  // _CRYPT_RAND_FP_H_
```

6.81 /tpm/include/private/prototypes/CryptRsa_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 03:18:00PM
 */

#ifndef _CRYPT_RSA_FP_H_
#define _CRYPT_RSA_FP_H_

#if ALG_RSA

//*** CryptRsaInit()
// Function called at _TPM_Init().
BOOL CryptRsaInit(void);

//*** CryptRsaStartup()
// Function called at TPM2_Startup()
BOOL CryptRsaStartup(void);

//*** CryptRsaPssSaltSize()
// This function computes the salt size used in PSS. It is broken out so that
// the X509 code can get the same value that is used by the encoding function in this
// module.
INT16
CryptRsaPssSaltSize(INT16 hashSize, INT16 outSize);
```

```
//*** MakeDerTag()
// Construct the DER value that is used in RSASSA
//  Return Type: INT16
//   > 0        size of value
//   <= 0       no hash exists
INT16
MakeDerTag(TPM_ALG_ID hashAlg, INT16 sizeOfBuffer, BYTE* buffer);

//*** CryptRsaSelectScheme()
// This function is used by TPM2_RSA_Decrypt and TPM2_RSA_Encrypt.  It sets up
// the rules to select a scheme between input and object default.
// This function assume the RSA object is loaded.
// If a default scheme is defined in object, the default scheme should be chosen,
// otherwise, the input scheme should be chosen.
// In the case that both the object and 'scheme' are not TPM_ALG_NULL, then
// if the schemes are the same, the input scheme will be chosen.
// if the scheme are not compatible, a NULL pointer will be returned.
//
// The return pointer may point to a TPM_ALG_NULL scheme.
TPMT_RSA_DECRYPT* CryptRsaSelectScheme(
    TPMI_DH_OBJECT    rsaHandle,  // IN: handle of an RSA key
    TPMT_RSA_DECRYPT* scheme      // IN: a sign or decrypt scheme
);

//*** CryptRsaLoadPrivateExponent()
// This function is called to generate the private exponent of an RSA key.
//  Return Type: TPM_RC
//      TPM_RC_BINDING      public and private parts of 'rsaKey' are not matched
TPM_RC
CryptRsaLoadPrivateExponent(TPMT_PUBLIC* publicArea, TPMT_SENSITIVE* sensitive);

//*** CryptRsaEncrypt()
// This is the entry point for encryption using RSA. Encryption is
// use of the public exponent. The padding parameter determines what
// padding will be used.
//
// The 'cOutSize' parameter must be at least as large as the size of the key.
//
// If the padding is RSA_PAD_NONE, 'dIn' is treated as a number. It must be
// lower in value than the key modulus.
// NOTE: If dIn has fewer bytes than cOut, then we don't add low-order zeros to
//       dIn to make it the size of the RSA key for the call to RSAEP. This is
//       because the high order bytes of dIn might have a numeric value that is
//       greater than the value of the key modulus. If this had low-order zeros
//       added, it would have a numeric value larger than the modulus even though
//       it started out with a lower numeric value.
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE     'cOutSize' is too small (must be the size
//                        of the modulus)
//      TPM_RC_SCHEME    'padType' is not a supported scheme
//
LIB_EXPORT TPM_RC CryptRsaEncrypt(
    TPM2B_PUBLIC_KEY_RSA* cOut,    // OUT: the encrypted data
    TPM2B*                dIn,     // IN: the data to encrypt
    OBJECT*               key,     // IN: the key used for encryption
    TPMT_RSA_DECRYPT*     scheme,  // IN: the type of padding and hash
                                   //     if needed
    const TPM2B* label,            // IN: in case it is needed
    RAND_STATE*  rand              // IN: random number generator
                                   //     state (mostly for testing)
);

//*** CryptRsaDecrypt()
// This is the entry point for decryption using RSA. Decryption is
// use of the private exponent. The 'padType' parameter determines what
```

```
// padding was used.
//
//  Return Type: TPM_RC
//      TPM_RC_SIZE         'cInSize' is not the same as the size of the public
//                          modulus of 'key'; or numeric value of the encrypted
//                          data is greater than the modulus
//      TPM_RC_VALUE        'dOutSize' is not large enough for the result
//      TPM_RC_SCHEME       'padType' is not supported
//
LIB_EXPORT TPM_RC CryptRsaDecrypt(
    TPM2B*             dOut,    // OUT: the decrypted data
    TPM2B*             cIn,     // IN: the data to decrypt
    OBJECT*            key,     // IN: the key to use for decryption
    TPMT_RSA_DECRYPT* scheme,   // IN: the padding scheme
    const TPM2B*       label    // IN: in case it is needed for the scheme
);

//*** CryptRsaSign()
// This function is used to generate an RSA signature of the type indicated in
// 'scheme'.
//
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       'scheme' or 'hashAlg' are not supported
//      TPM_RC_VALUE        'hInSize' does not match 'hashAlg' (for RSASSA)
//
LIB_EXPORT TPM_RC CryptRsaSign(TPMT_SIGNATURE* sigOut,
                               OBJECT*         key,  // IN: key to use
                               TPM2B_DIGEST*   hIn,  // IN: the digest to sign
                               RAND_STATE* rand  // IN: the random number generator
                                                 //      to use (mostly for testing)
);

//*** CryptRsaValidateSignature()
// This function is used to validate an RSA signature. If the signature is valid
// TPM_RC_SUCCESS is returned. If the signature is not valid, TPM_RC_SIGNATURE is
// returned. Other return codes indicate either parameter problems or fatal errors.
//
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE    the signature does not check
//      TPM_RC_SCHEME       unsupported scheme or hash algorithm
//
LIB_EXPORT TPM_RC CryptRsaValidateSignature(
    TPMT_SIGNATURE* sig,    // IN: signature
    OBJECT*         key,    // IN: public modulus
    TPM2B_DIGEST*   digest  // IN: The digest being validated
);

//*** CryptRsaGenerateKey()
// Generate an RSA key from a provided seed
//  Return Type: TPM_RC
//      TPM_RC_CANCELED     operation was canceled
//      TPM_RC_RANGE        public exponent is not supported
//      TPM_RC_VALUE        could not find a prime using the provided parameters
LIB_EXPORT TPM_RC CryptRsaGenerateKey(
    TPMT_PUBLIC*    publicArea,
    TPMT_SENSITIVE* sensitive,
    RAND_STATE*     rand  // IN: if not NULL, the deterministic
                          //     RNG state
);
#endif  // ALG_RSA

#endif  // _CRYPT_RSA_FP_H_
```

6.82 /tpm/include/private/prototypes/CryptSelfTest_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _CRYPT_SELF_TEST_FP_H_
#define _CRYPT_SELF_TEST_FP_H_

//*** CryptSelfTest()
// This function is called to start/complete a full self-test.
// If 'fullTest' is NO, then only the untested algorithms will be run. If
// 'fullTest' is YES, then 'g_untestedDecryptionAlgorithms' is reinitialized and then
// all tests are run.
// This implementation of the reference design does not support processing outside
// the framework of a TPM command. As a consequence, this command does not
// complete until all tests are done. Since this can take a long time, the TPM
// will check after each test to see if the command is canceled. If so, then the
// TPM will returned TPM_RC_CANCELLED. To continue with the self-tests, call
// TPM2_SelfTest(fullTest == No) and the TPM will complete the testing.
//  Return Type: TPM_RC
//      TPM_RC_CANCELED        if the command is canceled
LIB_EXPORT
TPM_RC
CryptSelfTest(TPMI_YES_NO fullTest  // IN: if full test is required
);

//*** CryptIncrementalSelfTest()
// This function is used to perform an incremental self-test. This implementation
// will perform the toTest values before returning. That is, it assumes that the
// TPM cannot perform background tasks between commands.
//
// This command may be canceled. If it is, then there is no return result.
// However, this command can be run again and the incremental progress will not
// be lost.
//  Return Type: TPM_RC
//      TPM_RC_CANCELED         processing of this command was canceled
//      TPM_RC_TESTING          if toTest list is not empty
//      TPM_RC_VALUE            an algorithm in the toTest list is not implemented
TPM_RC
CryptIncrementalSelfTest(TPML_ALG* toTest,   // IN: list of algorithms to be tested
                         TPML_ALG* toDoList  // OUT: list of algorithms needing test
);

//*** CryptInitializeToTest()
// This function will initialize the data structures for testing all the
// algorithms. This should not be called unless CryptAlgsSetImplemented() has
// been called
void CryptInitializeToTest(void);

//*** CryptTestAlgorithm()
// Only point of contact with the actual self tests. If a self-test fails, there
// is no return and the TPM goes into failure mode.
// The call to TestAlgorithm uses an algorithm selector and a bit vector. When the
// test is run, the corresponding bit in 'toTest' and in 'g_toTest' is CLEAR. If
// 'toTest' is NULL, then only the bit in 'g_toTest' is CLEAR.
// There is a special case for the call to TestAlgorithm(). When 'alg' is
// ALG_ERROR, TestAlgorithm() will CLEAR any bit in 'toTest' for which it has
// no test. This allows the knowledge about which algorithms have test to be
// accessed through the interface that provides the test.
//  Return Type: TPM_RC
//      TPM_RC_CANCELED     test was canceled
LIB_EXPORT
TPM_RC
CryptTestAlgorithm(TPM_ALG_ID alg, ALGORITHM_VECTOR* toTest);
```

```
#endif  // _CRYPT_SELF_TEST_FP_H_
```

6.83 /tpm/include/private/prototypes/CryptSmac_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _CRYPT_SMAC_FP_H_
#define _CRYPT_SMAC_FP_H_

#if SMAC_IMPLEMENTED

//*** CryptSmacStart()
// Function to start an SMAC.
UINT16
CryptSmacStart(HASH_STATE*         state,
               TPMU_PUBLIC_PARMS* keyParameters,
               TPM_ALG_ID          macAlg,  // IN: the type of MAC
               TPM2B*              key);

//*** CryptMacStart()
// Function to start either an HMAC or an SMAC. Cannot reuse the CryptHmacStart
// function because of the difference in number of parameters.
UINT16
CryptMacStart(HMAC_STATE*         state,
              TPMU_PUBLIC_PARMS* keyParameters,
              TPM_ALG_ID          macAlg,  // IN: the type of MAC
              TPM2B*              key);

//*** CryptMacEnd()
// Dispatch to the MAC end function using a size and buffer pointer.
UINT16
CryptMacEnd(HMAC_STATE* state, UINT32 size, BYTE* buffer);

//*** CryptMacEnd2B()
// Dispatch to the MAC end function using a 2B.
UINT16
CryptMacEnd2B(HMAC_STATE* state, TPM2B* data);
#endif  // SMAC_IMPLEMENTED

#endif  // _CRYPT_SMAC_FP_H_
```

6.84 /tpm/include/private/prototypes/CryptSym_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 03:18:00PM
 */

#ifndef _CRYPT_SYM_FP_H_
#define _CRYPT_SYM_FP_H_

//** Initialization and Data Access Functions
//
//*** CryptSymInit()
// This function is called to do _TPM_Init processing
BOOL CryptSymInit(void);

//*** CryptSymStartup()
// This function is called to do TPM2_Startup() processing
BOOL CryptSymStartup(void);
```

```
//*** CryptGetSymmetricBlockSize()
// This function returns the block size of the algorithm. The table of bit sizes has
// an entry for each allowed key size. The entry for a key size is 0 if the TPM does
// not implement that key size. The key size table is delimited with a negative number
// (-1). After the delimiter is a list of block sizes with each entry corresponding
// to the key bit size. For most symmetric algorithms, the block size is the same
// regardless of the key size but this arrangement allows them to be different.
//  Return Type: INT16
//   <= 0     cipher not supported
//   > 0      the cipher block size in bytes
LIB_EXPORT INT16 CryptGetSymmetricBlockSize(
    TPM_ALG_ID symmetricAlg,  // IN: the symmetric algorithm
    UINT16     keySizeInBits  // IN: the key size
);

//** Symmetric Encryption
// This function performs symmetric encryption based on the mode.
//  Return Type: TPM_RC
//      TPM_RC_SIZE         'dSize' is not a multiple of the block size for an
//                          algorithm that requires it
//      TPM_RC_FAILURE      Fatal error
LIB_EXPORT TPM_RC CryptSymmetricEncrypt(
    BYTE*       dOut,           // OUT:
    TPM_ALG_ID  algorithm,      // IN: the symmetric algorithm
    UINT16      keySizeInBits,  // IN: key size in bits
    const BYTE* key,            // IN: key buffer. The size of this buffer
                                //     in bytes is (keySizeInBits + 7) / 8
    TPM2B_IV*  ivInOut,         // IN/OUT: IV for decryption.
    TPM_ALG_ID mode,            // IN: Mode to use
    INT32      dSize,           // IN: data size (may need to be a
                                //     multiple of the blockSize)
    const BYTE* dIn             // IN: data buffer
);

//*** CryptSymmetricDecrypt()
// This function performs symmetric decryption based on the mode.
//  Return Type: TPM_RC
//      TPM_RC_FAILURE      A fatal error
//      TPM_RCS_SIZE        'dSize' is not a multiple of the block size for an
//                          algorithm that requires it
LIB_EXPORT TPM_RC CryptSymmetricDecrypt(
    BYTE*       dOut,           // OUT: decrypted data
    TPM_ALG_ID  algorithm,      // IN: the symmetric algorithm
    UINT16      keySizeInBits,  // IN: key size in bits
    const BYTE* key,            // IN: key buffer. The size of this buffer
                                //     in bytes is (keySizeInBits + 7) / 8
    TPM2B_IV*  ivInOut,         // IN/OUT: IV for decryption.
    TPM_ALG_ID mode,            // IN: Mode to use
    INT32      dSize,           // IN: data size (may need to be a
                                //     multiple of the blockSize)
    const BYTE* dIn             // IN: data buffer
);

//*** CryptSymKeyValidate()
// Validate that a provided symmetric key meets the requirements of the TPM
//  Return Type: TPM_RC
//      TPM_RC_KEY_SIZE         Key size specifiers do not match
//      TPM_RC_KEY              Key is not allowed
TPM_RC
CryptSymKeyValidate(TPMT_SYM_DEF_OBJECT* symDef, TPM2B_SYM_KEY* key);

#endif  // _CRYPT_SYM_FP_H_
```

6.85 /tpm/include/private/prototypes/CryptUtil_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Aug 30, 2019  Time: 02:11:54PM
 */

#ifndef _CRYPT_UTIL_FP_H_
#define _CRYPT_UTIL_FP_H_

//*** CryptIsSchemeAnonymous()
// This function is used to test a scheme to see if it is an anonymous scheme
// The only anonymous scheme is ECDAA. ECDAA can be used to do things
// like U-Prove.
BOOL CryptIsSchemeAnonymous(TPM_ALG_ID scheme  // IN: the scheme algorithm to test
);

//*** ParmDecryptSym()
//  This function performs parameter decryption using symmetric block cipher.
void ParmDecryptSym(TPM_ALG_ID symAlg,         // IN: the symmetric algorithm
                    TPM_ALG_ID hash,           // IN: hash algorithm for KDFa
                    UINT16     keySizeInBits,  // IN: the key size in bits
                    TPM2B*     key,            // IN: KDF HMAC key
                    TPM2B*     nonceCaller,    // IN: nonce caller
                    TPM2B*     nonceTpm,       // IN: nonce TPM
                    UINT32     dataSize,       // IN: size of parameter buffer
                    BYTE*      data            // OUT: buffer to be decrypted
);

//*** ParmEncryptSym()
//  This function performs parameter encryption using symmetric block cipher.
void ParmEncryptSym(TPM_ALG_ID symAlg,         // IN: symmetric algorithm
                    TPM_ALG_ID hash,           // IN: hash algorithm for KDFa
                    UINT16     keySizeInBits,  // IN: symmetric key size in bits
                    TPM2B*     key,            // IN: KDF HMAC key
                    TPM2B*     nonceCaller,    // IN: nonce caller
                    TPM2B*     nonceTpm,       // IN: nonce TPM
                    UINT32     dataSize,       // IN: size of parameter buffer
                    BYTE*      data            // OUT: buffer to be encrypted
);

//*** CryptXORObfuscation()
// This function implements XOR obfuscation. It should not be called if the
// hash algorithm is not implemented. The only return value from this function
// is TPM_RC_SUCCESS.
void CryptXORObfuscation(TPM_ALG_ID hash,      // IN: hash algorithm for KDF
                         TPM2B*     key,       // IN: KDF key
                         TPM2B*     contextU,  // IN: contextU
                         TPM2B*     contextV,  // IN: contextV
                         UINT32     dataSize,  // IN: size of data buffer
                         BYTE*      data       // IN/OUT: data to be XORed in place
);

//*** CryptInit()
// This function is called when the TPM receives a _TPM_Init indication.
//
// NOTE: The hash algorithms do not have to be tested, they just need to be
// available. They have to be tested before the TPM can accept HMAC authorization
// or return any result that relies on a hash algorithm.
//  Return Type: BOOL
//      TRUE(1)         initializations succeeded
//      FALSE(0)        initialization failed and caller should place the TPM into
//                      Failure Mode
BOOL CryptInit(void);

//*** CryptStartup()
```

```
// This function is called by TPM2_Startup() to initialize the functions in
// this cryptographic library and in the provided CryptoLibrary. This function
// and CryptUtilInit() are both provided so that the implementation may move the
// initialization around to get the best interaction.
//  Return Type: BOOL
//      TRUE(1)         startup succeeded
//      FALSE(0)        startup failed and caller should place the TPM into
//                      Failure Mode
BOOL CryptStartup(STARTUP_TYPE type  // IN: the startup type
);

//**********************************************************************
//** Algorithm-Independent Functions
//**********************************************************************
//*** Introduction
// These functions are used generically when a function of a general type
// (e.g., symmetric encryption) is required.  The functions will modify the
// parameters as required to interface to the indicated algorithms.
//
//*** CryptIsAsymAlgorithm()
// This function indicates if an algorithm is an asymmetric algorithm.
//  Return Type: BOOL
//      TRUE(1)         if it is an asymmetric algorithm
//      FALSE(0)        if it is not an asymmetric algorithm
BOOL CryptIsAsymAlgorithm(TPM_ALG_ID algID  // IN: algorithm ID
);

//*** CryptSecretEncrypt()
// This function creates a secret value and its associated secret structure using
// an asymmetric algorithm.
//
// This function is used by TPM2_Rewrap() TPM2_MakeCredential(),
// and TPM2_Duplicate().
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   'keyHandle' does not reference a valid decryption key
//      TPM_RC_KEY          invalid ECC key (public point is not on the curve)
//      TPM_RC_SCHEME       RSA key with an unsupported padding scheme
//      TPM_RC_VALUE        numeric value of the data to be decrypted is greater
//                          than the RSA key modulus
TPM_RC
CryptSecretEncrypt(OBJECT*      encryptKey,  // IN: encryption key object
                   const TPM2B* label,       // IN: a null-terminated string as L
                   TPM2B_DATA*  data,        // OUT: secret value
                   TPM2B_ENCRYPTED_SECRET* secret  // OUT: secret structure
);

//*** CryptSecretDecrypt()
// Decrypt a secret value by asymmetric (or symmetric) algorithm
// This function is used for ActivateCredential and Import for asymmetric
// decryption, and StartAuthSession for both asymmetric and symmetric
// decryption process
//
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       RSA key is not a decryption key
//      TPM_RC_BINDING          Invalid RSA key (public and private parts are not
//                              cryptographically bound.
//      TPM_RC_ECC_POINT        ECC point in the secret is not on the curve
//      TPM_RC_INSUFFICIENT     failed to retrieve ECC point from the secret
//      TPM_RC_NO_RESULT        multiplication resulted in ECC point at infinity
//      TPM_RC_SIZE             data to decrypt is not of the same size as RSA key
//      TPM_RC_VALUE            For RSA key, numeric value of the encrypted data is
//                              greater than the modulus, or the recovered data is
//                              larger than the output buffer.
//                              For keyedHash or symmetric key, the secret is
//                              larger than the size of the digest produced by
//                              the name algorithm.
```

```
//      TPM_RC_FAILURE           internal error
TPM_RC
CryptSecretDecrypt(OBJECT*      decryptKey,   // IN: decrypt key
                   TPM2B_NONCE* nonceCaller,  // IN: nonceCaller.  It is needed for
                                              //     symmetric decryption.  For
                                              //     asymmetric decryption, this
                                              //     parameter is NULL
                   const TPM2B*            label,   // IN: a value for L
                   TPM2B_ENCRYPTED_SECRET* secret,  // IN: input secret
                   TPM2B_DATA*             data     // OUT: decrypted secret value
);

//*** CryptParameterEncryption()
// This function does in-place encryption of a response parameter.
void CryptParameterEncryption(
    TPM_HANDLE handle,             // IN: encrypt session handle
    TPM2B*     nonceCaller,        // IN: nonce caller
    INT32      bufferSize,         // IN: size of parameter buffer
    UINT16     leadingSizeInByte,  // IN: the size of the leading size field in
                                   //     bytes
    TPM2B_AUTH* extraKey,          // IN: additional key material other than
                                   //     sessionAuth
    BYTE* buffer                   // IN/OUT: parameter buffer to be encrypted
);

//*** CryptParameterDecryption()
// This function does in-place decryption of a command parameter.
//  Return Type: TPM_RC
//      TPM_RC_SIZE             The number of bytes in the input buffer is less than
//                              the number of bytes to be decrypted.
TPM_RC
CryptParameterDecryption(
    TPM_HANDLE handle,             // IN: encrypted session handle
    TPM2B*     nonceCaller,        // IN: nonce caller
    INT32      bufferSize,         // IN: size of parameter buffer
    UINT16     leadingSizeInByte,  // IN: the size of the leading size field in
                                   //     byte
    TPM2B_AUTH* extraKey,          // IN: the authValue
    BYTE*       buffer             // IN/OUT: parameter buffer to be decrypted
);

//*** CryptComputeSymmetricUnique()
// This function computes the unique field in public area for symmetric objects.
void CryptComputeSymmetricUnique(
    TPMT_PUBLIC*    publicArea,  // IN: the object's public area
    TPMT_SENSITIVE* sensitive,   // IN: the associated sensitive area
    TPM2B_DIGEST*   unique       // OUT: unique buffer
);

//*** CryptCreateObject()
// This function creates an object.
// For an asymmetric key, it will create a key pair and, for a parent key, a seed
// value for child protections.
//
// For an symmetric object, (TPM_ALG_SYMCIPHER or TPM_ALG_KEYEDHASH), it will
// create a secret key if the caller did not provide one. It will create a random
// secret seed value that is hashed with the secret value to create the public
// unique value.
//
// 'publicArea', 'sensitive', and 'sensitiveCreate' are the only required parameters
// and are the only ones that are used by TPM2_Create(). The other parameters
// are optional and are used when the generated Object needs to be deterministic.
// This is the case for both Primary Objects and Derived Objects.
//
// When a seed value is provided, a RAND_STATE will be populated and used for
// all operations in the object generation that require a random number. In the
```

```
// simplest case, TPM2_CreatePrimary() will use 'seed', 'label' and 'context' with
// context being the hash of the template. If the Primary Object is in
// the Endorsement hierarchy, it will also populate 'proof' with ehProof.
//
// For derived keys, 'seed' will be the secret value from the parent, 'label' and
// 'context' will be set according to the parameters of TPM2_CreateLoaded() and
// 'hashAlg' will be set which causes the RAND_STATE to be a KDF generator.
//
//  Return Type: TPM_RC
//      TPM_RC_KEY          a provided key is not an allowed value
//      TPM_RC_KEY_SIZE     key size in the public area does not match the size
//                          in the sensitive creation area for a symmetric key
//      TPM_RC_NO_RESULT    unable to get random values (only in derivation)
//      TPM_RC_RANGE        for an RSA key, the exponent is not supported
//      TPM_RC_SIZE         sensitive data size is larger than allowed for the
//                          scheme for a keyed hash object
//      TPM_RC_VALUE        exponent is not prime or could not find a prime using
//                          the provided parameters for an RSA key;
//                          unsupported name algorithm for an ECC key
TPM_RC
CryptCreateObject(OBJECT*                object,  // IN: new object structure pointer
                  TPMS_SENSITIVE_CREATE* sensitiveCreate,  // IN: sensitive creation
                  RAND_STATE*            rand  // IN: the random number generator
                                               //      to use
);

//*** CryptGetSignHashAlg()
// Get the hash algorithm of signature from a TPMT_SIGNATURE structure.
// It assumes the signature is not NULL
//  This is a function for easy access
TPMI_ALG_HASH
CryptGetSignHashAlg(TPMT_SIGNATURE* auth  // IN: signature
);

//*** CryptIsSplitSign()
// This function us used to determine if the signing operation is a split
// signing operation that required a TPM2_Commit().
//
BOOL CryptIsSplitSign(TPM_ALG_ID scheme  // IN: the algorithm selector
);

//*** CryptIsAsymSignScheme()
// This function indicates if a scheme algorithm is a sign algorithm.
BOOL CryptIsAsymSignScheme(TPMI_ALG_PUBLIC      publicType,  // IN: Type of the object
                           TPMI_ALG_ASYM_SCHEME scheme       // IN: the scheme
);

//*** CryptIsAsymDecryptScheme()
// This function indicate if a scheme algorithm is a decrypt algorithm.
BOOL CryptIsAsymDecryptScheme(TPMI_ALG_PUBLIC publicType,  // IN: Type of the object
                              TPMI_ALG_ASYM_SCHEME scheme  // IN: the scheme
);

//*** CryptSelectSignScheme()
// This function is used by the attestation and signing commands.  It implements
// the rules for selecting the signature scheme to use in signing. This function
// requires that the signing key either be TPM_RH_NULL or be loaded.
//
// If a default scheme is defined in object, the default scheme should be chosen,
// otherwise, the input scheme should be chosen.
// In the case that  both object and input scheme has a non-NULL scheme
// algorithm, if the schemes are compatible, the input scheme will be chosen.
//
// This function should not be called if 'signObject->publicArea.type' ==
// ALG_SYMCIPHER.
//
```

```
//  Return Type: BOOL
//      TRUE(1)         scheme selected
//      FALSE(0)        both 'scheme' and key's default scheme are empty; or
//                      'scheme' is empty while key's default scheme requires
//                      explicit input scheme (split signing); or
//                      non-empty default key scheme differs from 'scheme'
BOOL CryptSelectSignScheme(OBJECT*          signObject,  // IN: signing key
                           TPMT_SIG_SCHEME* scheme       // IN/OUT: signing scheme
);

//*** CryptSign()
// Sign a digest with asymmetric key or HMAC.
// This function is called by attestation commands and the generic TPM2_Sign
// command.
// This function checks the key scheme and digest size.  It does not
// check if the sign operation is allowed for restricted key.  It should be
// checked before the function is called.
// The function will assert if the key is not a signing key.
//
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       'signScheme' is not compatible with the signing key type
//      TPM_RC_VALUE        'digest' value is greater than the modulus of
//                          'signHandle' or size of 'hashData' does not match hash
//                          algorithm in'signScheme' (for an RSA key);
//                          invalid commit status or failed to generate "r" value
//                          (for an ECC key)
TPM_RC
CryptSign(OBJECT*           signKey,     // IN: signing key
          TPMT_SIG_SCHEME*  signScheme,  // IN: sign scheme.
          TPM2B_DIGEST*     digest,      // IN: The digest being signed
          TPMT_SIGNATURE*   signature    // OUT: signature
);

//*** CryptValidateSignature()
// This function is used to verify a signature.  It is called by
// TPM2_VerifySignature() and TPM2_PolicySigned.
//
// Since this operation only requires use of a public key, no consistency
// checks are necessary for the key to signature type because a caller can load
// any public key that they like with any scheme that they like. This routine
// simply makes sure that the signature is correct, whatever the type.
//
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE            the signature is not genuine
//      TPM_RC_SCHEME               the scheme is not supported
//      TPM_RC_HANDLE               an HMAC key was selected but the
//                                  private part of the key is not loaded
TPM_RC
CryptValidateSignature(TPMI_DH_OBJECT  keyHandle,  // IN: The handle of sign key
                       TPM2B_DIGEST*   digest,     // IN: The digest being validated
                       TPMT_SIGNATURE* signature   // IN: signature
);

//*** CryptGetTestResult
// This function returns the results of a self-test function.
// Note: the behavior in this function is NOT the correct behavior for a real
// TPM implementation.  An artificial behavior is placed here due to the
// limitation of a software simulation environment.  For the correct behavior,
// consult the part 3 specification for TPM2_GetTestResult().
TPM_RC
CryptGetTestResult(TPM2B_MAX_BUFFER* outData  // OUT: test result data
);

//*** CryptValidateKeys()
// This function is used to verify that the key material of and object is valid.
// For a 'publicOnly' object, the key is verified for size and, if it is an ECC
```

```
// key, it is verified to be on the specified curve. For a key with a sensitive
// area, the binding between the public and private parts of the key are verified.
// If the nameAlg of the key is TPM_ALG_NULL, then the size of the sensitive area
// is verified but the public portion is not verified, unless the key is an RSA key.
// For an RSA key, the reason for loading the sensitive area is to use it. The
// only way to use a private RSA key is to compute the private exponent. To compute
// the private exponent, the public modulus is used.
//  Return Type: TPM_RC
//      TPM_RC_BINDING      the public and private parts are not cryptographically
//                          bound
//      TPM_RC_HASH         cannot have a publicOnly key with nameAlg of TPM_ALG_NULL
//      TPM_RC_KEY          the public unique is not valid
//      TPM_RC_KEY_SIZE     the private area key is not valid
//      TPM_RC_TYPE         the types of the sensitive and private parts do not match
TPM_RC
CryptValidateKeys(TPMT_PUBLIC*    publicArea,
                  TPMT_SENSITIVE* sensitive,
                  TPM_RC          blamePublic,
                  TPM_RC          blameSensitive);

//*** CryptSelectMac()
// This function is used to set the MAC scheme based on the key parameters and
// the input scheme.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       the scheme is not a valid mac scheme
//      TPM_RC_TYPE         the input key is not a type that supports a mac
//      TPM_RC_VALUE        the input scheme and the key scheme are not compatible
TPM_RC
CryptSelectMac(TPMT_PUBLIC* publicArea, TPMI_ALG_MAC_SCHEME* inMac);

//*** CryptMacIsValidForKey()
// Check to see if the key type is compatible with the mac type
BOOL CryptMacIsValidForKey(TPM_ALG_ID keyType, TPM_ALG_ID macAlg, BOOL flag);

//*** CryptSmacIsValidAlg()
// This function is used to test if an algorithm is a supported SMAC algorithm. It
// needs to be updated as new algorithms are added.
BOOL CryptSmacIsValidAlg(TPM_ALG_ID alg,
                         BOOL       FLAG  // IN: Indicates if TPM_ALG_NULL is valid
);

//*** CryptSymModeIsValid()
// Function checks to see if an algorithm ID is a valid, symmetric block cipher
// mode for the TPM. If 'flag' is SET, them TPM_ALG_NULL is a valid mode.
// not include the modes used for SMAC
BOOL CryptSymModeIsValid(TPM_ALG_ID mode, BOOL flag);

#endif  // _CRYPT_UTIL_FP_H_
```

6.86 /tpm/include/private/prototypes/DA_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 04:23:27PM
 */

#ifndef _DA_FP_H_
#define _DA_FP_H_

//*** DAPreInstall_Init()
// This function initializes the DA parameters to their manufacturer-default
// values. The default values are determined by a platform-specific specification.
//
// This function should not be called outside of a manufacturing or simulation
// environment.
```

```
//
// The DA parameters will be restored to these initial values by TPM2_Clear().
void DAPreInstall_Init(void);

//*** DAStartup()
// This function is called  by TPM2_Startup() to initialize the DA parameters.
// In the case of Startup(CLEAR), use of lockoutAuth will be enabled if the
// lockout recovery time is 0. Otherwise, lockoutAuth will not be enabled until
// the TPM has been continuously powered for the lockoutRecovery time.
//
// This function requires that NV be available and not rate limiting.
BOOL DAStartup(STARTUP_TYPE type  // IN: startup type
);

//*** DARegisterFailure()
// This function is called when a authorization failure occurs on an entity
// that is subject to dictionary-attack protection. When a DA failure is
// triggered, register the failure by resetting the relevant self-healing
// timer to the current time.
void DARegisterFailure(TPM_HANDLE handle  // IN: handle for failure
);

//*** DASelfHeal()
// This function is called to check if sufficient time has passed to allow
// decrement of failedTries or to re-enable use of lockoutAuth.
//
// This function should be called when the time interval is updated.
void DASelfHeal(void);

#endif  // _DA_FP_H_
```

6.87 /tpm/include/private/prototypes/DictionaryAttackLockReset_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_DictionaryAttackLockReset  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_DICTIONARYATTACKLOCKRESET_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_DICTIONARYATTACKLOCKRESET_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_LOCKOUT lockHandle;
} DictionaryAttackLockReset_In;

// Response code modifiers
#    define RC_DictionaryAttackLockReset_lockHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_DictionaryAttackLockReset(DictionaryAttackLockReset_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_DICTIONARYATTACKLOCKRESET_FP_H_
#endif    // CC_DictionaryAttackLockReset
```

6.88 /tpm/include/private/prototypes/DictionaryAttackParameters_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_DictionaryAttackParameters  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_DICTIONARYATTACKPARAMETERS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_DICTIONARYATTACKPARAMETERS_FP_H_
```

```
// Input structure definition
typedef struct
{
    TPMI_RH_LOCKOUT lockHandle;
    UINT32          newMaxTries;
    UINT32          newRecoveryTime;
    UINT32          lockoutRecovery;
} DictionaryAttackParameters_In;

// Response code modifiers
#    define RC_DictionaryAttackParameters_lockHandle      (TPM_RC_H + TPM_RC_1)
#    define RC_DictionaryAttackParameters_newMaxTries     (TPM_RC_P + TPM_RC_1)
#    define RC_DictionaryAttackParameters_newRecoveryTime (TPM_RC_P + TPM_RC_2)
#    define RC_DictionaryAttackParameters_lockoutRecovery (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_DictionaryAttackParameters(DictionaryAttackParameters_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_DICTIONARYATTACKPARAMETERS_FP_H_
#endif    // CC_DictionaryAttackParameters
```

6.89 /tpm/include/private/prototypes/Duplicate_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Duplicate  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_DUPLICATE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_DUPLICATE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT      objectHandle;
    TPMI_DH_OBJECT      newParentHandle;
    TPM2B_DATA          encryptionKeyIn;
    TPMT_SYM_DEF_OBJECT symmetricAlg;
} Duplicate_In;

// Output structure definition
typedef struct
{
    TPM2B_DATA             encryptionKeyOut;
    TPM2B_PRIVATE          duplicate;
    TPM2B_ENCRYPTED_SECRET outSymSeed;
} Duplicate_Out;

// Response code modifiers
#    define RC_Duplicate_objectHandle    (TPM_RC_H + TPM_RC_1)
#    define RC_Duplicate_newParentHandle (TPM_RC_H + TPM_RC_2)
#    define RC_Duplicate_encryptionKeyIn (TPM_RC_P + TPM_RC_1)
#    define RC_Duplicate_symmetricAlg    (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_Duplicate(Duplicate_In* in, Duplicate_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_DUPLICATE_FP_H_
#endif    // CC_Duplicate
```

6.90 /tpm/include/private/prototypes/ECC_Decrypt_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ECC_Decrypt  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_DECRYPT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_DECRYPT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   keyHandle;
    TPM2B_ECC_POINT  C1;
    TPM2B_MAX_BUFFER C2;
    TPM2B_DIGEST     C3;
    TPMT_KDF_SCHEME  inScheme;
} ECC_Decrypt_In;

// Output structure definition
typedef struct
{
    TPM2B_MAX_BUFFER plainText;
} ECC_Decrypt_Out;

// Response code modifiers
#    define RC_ECC_Decrypt_keyHandle (TPM_RC_H + TPM_RC_1)
#    define RC_ECC_Decrypt_C1        (TPM_RC_P + TPM_RC_1)
#    define RC_ECC_Decrypt_C2        (TPM_RC_P + TPM_RC_2)
#    define RC_ECC_Decrypt_C3        (TPM_RC_P + TPM_RC_3)
#    define RC_ECC_Decrypt_inScheme  (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_ECC_Decrypt(ECC_Decrypt_In* in, ECC_Decrypt_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_DECRYPT_FP_H_
#endif    // CC_ECC_Decrypt
```

6.91 /tpm/include/private/prototypes/ECC_Encrypt_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ECC_Encrypt  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_ENCRYPT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_ENCRYPT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   keyHandle;
    TPM2B_MAX_BUFFER plainText;
    TPMT_KDF_SCHEME  inScheme;
} ECC_Encrypt_In;

// Output structure definition
typedef struct
{
    TPM2B_ECC_POINT  C1;
    TPM2B_MAX_BUFFER C2;
    TPM2B_DIGEST     C3;
} ECC_Encrypt_Out;

// Response code modifiers
```

```
#    define RC_ECC_Encrypt_keyHandle (TPM_RC_H + TPM_RC_1)
#    define RC_ECC_Encrypt_plainText (TPM_RC_P + TPM_RC_1)
#    define RC_ECC_Encrypt_inScheme  (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_ECC_Encrypt(ECC_Encrypt_In* in, ECC_Encrypt_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_ENCRYPT_FP_H_
#endif    // CC_ECC_Encrypt
```

6.92 /tpm/include/private/prototypes/ECC_Parameters_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ECC_Parameters  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_PARAMETERS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_PARAMETERS_FP_H_

// Input structure definition
typedef struct
{
    TPMI_ECC_CURVE curveID;
} ECC_Parameters_In;

// Output structure definition
typedef struct
{
    TPMS_ALGORITHM_DETAIL_ECC parameters;
} ECC_Parameters_Out;

// Response code modifiers
#    define RC_ECC_Parameters_curveID (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ECC_Parameters(ECC_Parameters_In* in, ECC_Parameters_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECC_PARAMETERS_FP_H_
#endif    // CC_ECC_Parameters
```

6.93 /tpm/include/private/prototypes/ECDH_KeyGen_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ECDH_KeyGen  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECDH_KEYGEN_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECDH_KEYGEN_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT keyHandle;
} ECDH_KeyGen_In;

// Output structure definition
typedef struct
{
    TPM2B_ECC_POINT zPoint;
    TPM2B_ECC_POINT pubPoint;
} ECDH_KeyGen_Out;
```

```
// Response code modifiers
#    define RC_ECDH_KeyGen_keyHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ECDH_KeyGen(ECDH_KeyGen_In* in, ECDH_KeyGen_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECDH_KEYGEN_FP_H_
#endif    // CC_ECDH_KeyGen
```

6.94 /tpm/include/private/prototypes/ECDH_ZGen_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ECDH_ZGen  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECDH_ZGEN_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECDH_ZGEN_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT  keyHandle;
    TPM2B_ECC_POINT inPoint;
} ECDH_ZGen_In;

// Output structure definition
typedef struct
{
    TPM2B_ECC_POINT outPoint;
} ECDH_ZGen_Out;

// Response code modifiers
#    define RC_ECDH_ZGen_keyHandle (TPM_RC_H + TPM_RC_1)
#    define RC_ECDH_ZGen_inPoint   (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ECDH_ZGen(ECDH_ZGen_In* in, ECDH_ZGen_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ECDH_ZGEN_FP_H_
#endif    // CC_ECDH_ZGen
```

6.95 /tpm/include/private/prototypes/EC_Ephemeral_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_EC_Ephemeral  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_EC_EPHEMERAL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_EC_EPHEMERAL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_ECC_CURVE curveID;
} EC_Ephemeral_In;

// Output structure definition
typedef struct
{
    TPM2B_ECC_POINT Q;
    UINT16          counter;
} EC_Ephemeral_Out;
```

```
// Response code modifiers
#    define RC_EC_Ephemeral_curveID (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_EC_Ephemeral(EC_Ephemeral_In* in, EC_Ephemeral_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_EC_EPHEMERAL_FP_H_
#endif    // CC_EC_Ephemeral
```

6.96 /tpm/include/private/prototypes/EncryptDecrypt2_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_EncryptDecrypt2  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ENCRYPTDECRYPT2_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ENCRYPTDECRYPT2_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT       keyHandle;
    TPM2B_MAX_BUFFER     inData;
    TPMI_YES_NO          decrypt;
    TPMI_ALG_CIPHER_MODE mode;
    TPM2B_IV             ivIn;
} EncryptDecrypt2_In;

// Output structure definition
typedef struct
{
    TPM2B_MAX_BUFFER outData;
    TPM2B_IV         ivOut;
} EncryptDecrypt2_Out;

// Response code modifiers
#    define RC_EncryptDecrypt2_keyHandle (TPM_RC_H + TPM_RC_1)
#    define RC_EncryptDecrypt2_inData    (TPM_RC_P + TPM_RC_1)
#    define RC_EncryptDecrypt2_decrypt   (TPM_RC_P + TPM_RC_2)
#    define RC_EncryptDecrypt2_mode      (TPM_RC_P + TPM_RC_3)
#    define RC_EncryptDecrypt2_ivIn      (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_EncryptDecrypt2(EncryptDecrypt2_In* in, EncryptDecrypt2_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ENCRYPTDECRYPT2_FP_H_
#endif    // CC_EncryptDecrypt2
```

6.97 /tpm/include/private/prototypes/EncryptDecrypt_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_EncryptDecrypt  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ENCRYPTDECRYPT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ENCRYPTDECRYPT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT       keyHandle;
```

```
    TPMI_YES_NO          decrypt;
    TPMI_ALG_CIPHER_MODE mode;
    TPM2B_IV             ivIn;
    TPM2B_MAX_BUFFER     inData;
} EncryptDecrypt_In;

// Output structure definition
typedef struct
{
    TPM2B_MAX_BUFFER outData;
    TPM2B_IV         ivOut;
} EncryptDecrypt_Out;

// Response code modifiers
#    define RC_EncryptDecrypt_keyHandle (TPM_RC_H + TPM_RC_1)
#    define RC_EncryptDecrypt_decrypt   (TPM_RC_P + TPM_RC_1)
#    define RC_EncryptDecrypt_mode      (TPM_RC_P + TPM_RC_2)
#    define RC_EncryptDecrypt_ivIn      (TPM_RC_P + TPM_RC_3)
#    define RC_EncryptDecrypt_inData    (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_EncryptDecrypt(EncryptDecrypt_In* in, EncryptDecrypt_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ENCRYPTDECRYPT_FP_H_
#endif    // CC_EncryptDecrypt
```

6.98 /tpm/include/private/prototypes/EncryptDecrypt_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _ENCRYPT_DECRYPT_SPT_FP_H_
#define _ENCRYPT_DECRYPT_SPT_FP_H_

#if CC_EncryptDecrypt2

//  Return Type: TPM_RC
//      TPM_RC_KEY          is not a symmetric decryption key with both
//                          public and private portions loaded
//      TPM_RC_SIZE         'IvIn' size is incompatible with the block cipher mode;
//                          or 'inData' size is not an even multiple of the block
//                          size for CBC or ECB mode
//      TPM_RC_VALUE        'keyHandle' is restricted and the argument 'mode' does
//                          not match the key's mode
TPM_RC
EncryptDecryptShared(TPMI_DH_OBJECT      keyHandleIn,
                     TPMI_YES_NO         decryptIn,
                     TPMI_ALG_SYM_MODE   modeIn,
                     TPM2B_IV*           ivIn,
                     TPM2B_MAX_BUFFER*   inData,
                     EncryptDecrypt_Out* out);
#endif  // CC_EncryptDecrypt

#endif  // _ENCRYPT_DECRYPT_SPT_FP_H_
```

6.99 /tpm/include/private/prototypes/Entity_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  7, 2020  Time: 07:19:36PM
 */
```

```
#ifndef _ENTITY_FP_H_
#define _ENTITY_FP_H_

//** Functions
//*** EntityGetLoadStatus()
// This function will check that all the handles access loaded entities.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE           handle type does not match
//      TPM_RC_REFERENCE_Hx     entity is not present
//      TPM_RC_HIERARCHY        entity belongs to a disabled hierarchy
//      TPM_RC_OBJECT_MEMORY    handle is an evict object but there is no
//                               space to load it to RAM
TPM_RC
EntityGetLoadStatus(COMMAND* command  // IN/OUT: command parsing structure
);

//*** EntityGetAuthValue()
// This function is used to access the 'authValue' associated with a handle.
// This function assumes that the handle references an entity that is accessible
// and the handle is not for a persistent objects. That is EntityGetLoadStatus()
// should have been called. Also, the accessibility of the authValue should have
// been verified by IsAuthValueAvailable().
//
// This function copies the authorization value of the entity to 'auth'.
// Return Type: UINT16
//      count           number of bytes in the authValue with 0's stripped
UINT16
EntityGetAuthValue(TPMI_DH_ENTITY handle,  // IN: handle of entity
                   TPM2B_AUTH*    auth     // OUT: authValue of the entity
);

//*** EntityGetAuthPolicy()
// This function is used to access the 'authPolicy' associated with a handle.
// This function assumes that the handle references an entity that is accessible
// and the handle is not for a persistent objects. That is EntityGetLoadStatus()
// should have been called. Also, the accessibility of the authPolicy should have
// been verified by IsAuthPolicyAvailable().
//
// This function copies the authorization policy of the entity to 'authPolicy'.
//
//  The return value is the hash algorithm for the policy.
TPMI_ALG_HASH
EntityGetAuthPolicy(TPMI_DH_ENTITY handle,     // IN: handle of entity
                    TPM2B_DIGEST*  authPolicy  // OUT: authPolicy of the entity
);

//*** EntityGetName()
// This function returns the Name associated with a handle.
TPM2B_NAME* EntityGetName(TPMI_DH_ENTITY handle,  // IN: handle of entity
                          TPM2B_NAME*    name     // OUT: name of entity
);

//*** EntityGetHierarchy()
// This function returns the hierarchy handle associated with an entity.
// a) A handle that is a hierarchy handle is associated with itself.
// b) An NV index belongs to TPM_RH_PLATFORM if TPMA_NV_PLATFORMCREATE,
//    is SET, otherwise it belongs to TPM_RH_OWNER
// c) An object handle belongs to its hierarchy.
TPMI_RH_HIERARCHY
EntityGetHierarchy(TPMI_DH_ENTITY handle  // IN :handle of entity
);

#endif  // _ENTITY_FP_H_
```

6.100 /tpm/include/private/prototypes/EventSequenceComplete_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_EventSequenceComplete  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_EVENTSEQUENCECOMPLETE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_EVENTSEQUENCECOMPLETE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_PCR      pcrHandle;
    TPMI_DH_OBJECT   sequenceHandle;
    TPM2B_MAX_BUFFER buffer;
} EventSequenceComplete_In;

// Output structure definition
typedef struct
{
    TPML_DIGEST_VALUES results;
} EventSequenceComplete_Out;

// Response code modifiers
#    define RC_EventSequenceComplete_pcrHandle      (TPM_RC_H + TPM_RC_1)
#    define RC_EventSequenceComplete_sequenceHandle (TPM_RC_H + TPM_RC_2)
#    define RC_EventSequenceComplete_buffer         (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_EventSequenceComplete(EventSequenceComplete_In*  in,
                           EventSequenceComplete_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_EVENTSEQUENCECOMPLETE_FP_H_
#endif    // CC_EventSequenceComplete
```

6.101 /tpm/include/private/prototypes/EvictControl_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_EvictControl  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_EVICTCONTROL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_EVICTCONTROL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION  auth;
    TPMI_DH_OBJECT     objectHandle;
    TPMI_DH_PERSISTENT persistentHandle;
} EvictControl_In;

// Response code modifiers
#    define RC_EvictControl_auth             (TPM_RC_H + TPM_RC_1)
#    define RC_EvictControl_objectHandle     (TPM_RC_H + TPM_RC_2)
#    define RC_EvictControl_persistentHandle (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_EvictControl(EvictControl_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_EVICTCONTROL_FP_H_
#endif    // CC_EvictControl
```

6.102 /tpm/include/private/prototypes/FieldUpgradeData_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_FieldUpgradeData  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIELDUPGRADEDATA_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIELDUPGRADEDATA_FP_H_

// Input structure definition
typedef struct
{
    TPM2B_MAX_BUFFER fuData;
} FieldUpgradeData_In;

// Output structure definition
typedef struct
{
    TPMT_HA nextDigest;
    TPMT_HA firstDigest;
} FieldUpgradeData_Out;

// Response code modifiers
#    define RC_FieldUpgradeData_fuData (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_FieldUpgradeData(FieldUpgradeData_In* in, FieldUpgradeData_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIELDUPGRADEDATA_FP_H_
#endif    // CC_FieldUpgradeData
```

6.103 /tpm/include/private/prototypes/FieldUpgradeStart_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_FieldUpgradeStart  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIELDUPGRADESTART_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIELDUPGRADESTART_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PLATFORM authorization;
    TPMI_DH_OBJECT   keyHandle;
    TPM2B_DIGEST     fuDigest;
    TPMT_SIGNATURE   manifestSignature;
} FieldUpgradeStart_In;

// Response code modifiers
#    define RC_FieldUpgradeStart_authorization     (TPM_RC_H + TPM_RC_1)
#    define RC_FieldUpgradeStart_keyHandle         (TPM_RC_H + TPM_RC_2)
#    define RC_FieldUpgradeStart_fuDigest          (TPM_RC_P + TPM_RC_1)
#    define RC_FieldUpgradeStart_manifestSignature (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_FieldUpgradeStart(FieldUpgradeStart_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIELDUPGRADESTART_FP_H_
#endif    // CC_FieldUpgradeStart
```

6.104 /tpm/include/private/prototypes/FirmwareRead_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_FirmwareRead  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIRMWAREREAD_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIRMWAREREAD_FP_H_

// Input structure definition
typedef struct
{
    UINT32 sequenceNumber;
} FirmwareRead_In;

// Output structure definition
typedef struct
{
    TPM2B_MAX_BUFFER fuData;
} FirmwareRead_Out;

// Response code modifiers
#    define RC_FirmwareRead_sequenceNumber (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_FirmwareRead(FirmwareRead_In* in, FirmwareRead_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_FIRMWAREREAD_FP_H_
#endif    // CC_FirmwareRead
```

6.105 /tpm/include/private/prototypes/FlushContext_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_FlushContext  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_FLUSHCONTEXT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_FLUSHCONTEXT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_CONTEXT flushHandle;
} FlushContext_In;

// Response code modifiers
#    define RC_FlushContext_flushHandle (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_FlushContext(FlushContext_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_FLUSHCONTEXT_FP_H_
#endif    // CC_FlushContext
```

6.106 /tpm/include/private/prototypes/GetCapability_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_GetCapability  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETCAPABILITY_FP_H_
```

```
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETCAPABILITY_FP_H_

// Input structure definition
typedef struct
{
    TPM_CAP capability;
    UINT32  property;
    UINT32  propertyCount;
} GetCapability_In;

// Output structure definition
typedef struct
{
    TPMI_YES_NO          moreData;
    TPMS_CAPABILITY_DATA capabilityData;
} GetCapability_Out;

// Response code modifiers
#    define RC_GetCapability_capability     (TPM_RC_P + TPM_RC_1)
#    define RC_GetCapability_property       (TPM_RC_P + TPM_RC_2)
#    define RC_GetCapability_propertyCount (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_GetCapability(GetCapability_In* in, GetCapability_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETCAPABILITY_FP_H_
#endif    // CC_GetCapability
```

6.107 /tpm/include/private/prototypes/GetCommandAuditDigest_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_GetCommandAuditDigest  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETCOMMANDAUDITDIGEST_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETCOMMANDAUDITDIGEST_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_ENDORSEMENT privacyHandle;
    TPMI_DH_OBJECT      signHandle;
    TPM2B_DATA          qualifyingData;
    TPMT_SIG_SCHEME     inScheme;
} GetCommandAuditDigest_In;

// Output structure definition
typedef struct
{
    TPM2B_ATTEST   auditInfo;
    TPMT_SIGNATURE signature;
} GetCommandAuditDigest_Out;

// Response code modifiers
#    define RC_GetCommandAuditDigest_privacyHandle  (TPM_RC_H + TPM_RC_1)
#    define RC_GetCommandAuditDigest_signHandle     (TPM_RC_H + TPM_RC_2)
#    define RC_GetCommandAuditDigest_qualifyingData (TPM_RC_P + TPM_RC_1)
#    define RC_GetCommandAuditDigest_inScheme       (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_GetCommandAuditDigest(GetCommandAuditDigest_In*  in,
                           GetCommandAuditDigest_Out* out);
```

```
#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETCOMMANDAUDITDIGEST_FP_H_
#endif    // CC_GetCommandAuditDigest
```

6.108 /tpm/include/private/prototypes/GetRandom_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_GetRandom  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETRANDOM_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETRANDOM_FP_H_

// Input structure definition
typedef struct
{
    UINT16 bytesRequested;
} GetRandom_In;

// Output structure definition
typedef struct
{
    TPM2B_DIGEST randomBytes;
} GetRandom_Out;

// Response code modifiers
#    define RC_GetRandom_bytesRequested (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_GetRandom(GetRandom_In* in, GetRandom_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETRANDOM_FP_H_
#endif    // CC_GetRandom
```

6.109 /tpm/include/private/prototypes/GetSessionAuditDigest_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_GetSessionAuditDigest  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETSESSIONAUDITDIGEST_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETSESSIONAUDITDIGEST_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_ENDORSEMENT privacyAdminHandle;
    TPMI_DH_OBJECT      signHandle;
    TPMI_SH_HMAC        sessionHandle;
    TPM2B_DATA          qualifyingData;
    TPMT_SIG_SCHEME     inScheme;
} GetSessionAuditDigest_In;

// Output structure definition
typedef struct
{
    TPM2B_ATTEST   auditInfo;
    TPMT_SIGNATURE signature;
} GetSessionAuditDigest_Out;

// Response code modifiers
#    define RC_GetSessionAuditDigest_privacyAdminHandle (TPM_RC_H + TPM_RC_1)
#    define RC_GetSessionAuditDigest_signHandle         (TPM_RC_H + TPM_RC_2)
#    define RC_GetSessionAuditDigest_sessionHandle      (TPM_RC_H + TPM_RC_3)
```

```
#    define RC_GetSessionAuditDigest_qualifyingData     (TPM_RC_P + TPM_RC_1)
#    define RC_GetSessionAuditDigest_inScheme           (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_GetSessionAuditDigest(GetSessionAuditDigest_In*  in,
                           GetSessionAuditDigest_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETSESSIONAUDITDIGEST_FP_H_
#endif    // CC_GetSessionAuditDigest
```

6.110 /tpm/include/private/prototypes/GetTestResult_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_GetTestResult  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETTESTRESULT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETTESTRESULT_FP_H_

// Output structure definition
typedef struct
{
    TPM2B_MAX_BUFFER outData;
    TPM_RC           testResult;
} GetTestResult_Out;

// Function prototype
TPM_RC
TPM2_GetTestResult(GetTestResult_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETTESTRESULT_FP_H_
#endif    // CC_GetTestResult
```

6.111 /tpm/include/private/prototypes/GetTime_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_GetTime  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETTIME_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETTIME_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_ENDORSEMENT privacyAdminHandle;
    TPMI_DH_OBJECT      signHandle;
    TPM2B_DATA          qualifyingData;
    TPMT_SIG_SCHEME     inScheme;
} GetTime_In;

// Output structure definition
typedef struct
{
    TPM2B_ATTEST   timeInfo;
    TPMT_SIGNATURE signature;
} GetTime_Out;

// Response code modifiers
#    define RC_GetTime_privacyAdminHandle (TPM_RC_H + TPM_RC_1)
#    define RC_GetTime_signHandle         (TPM_RC_H + TPM_RC_2)
#    define RC_GetTime_qualifyingData     (TPM_RC_P + TPM_RC_1)
#    define RC_GetTime_inScheme           (TPM_RC_P + TPM_RC_2)
```

```
// Function prototype
TPM_RC
TPM2_GetTime(GetTime_In* in, GetTime_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_GETTIME_FP_H_
#endif    // CC_GetTime
```

6.112 /tpm/include/private/prototypes/Handle_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _HANDLE_FP_H_
#define _HANDLE_FP_H_

//*** HandleGetType()
// This function returns the type of a handle which is the MSO of the handle.
TPM_HT
HandleGetType(TPM_HANDLE handle  // IN: a handle to be checked
);

//*** NextPermanentHandle()
// This function returns the permanent handle that is equal to the input value or
// is the next higher value. If there is no handle with the input value and there
// is no next higher value, it returns 0:
TPM_HANDLE
NextPermanentHandle(TPM_HANDLE inHandle  // IN: the handle to check
);

//*** PermanentCapGetHandles()
// This function returns a list of the permanent handles of PCR, started from
// 'handle'. If 'handle' is larger than the largest permanent handle, an empty list
// will be returned with 'more' set to NO.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
PermanentCapGetHandles(TPM_HANDLE   handle,     // IN: start handle
                       UINT32       count,      // IN: count of returned handles
                       TPML_HANDLE* handleList  // OUT: list of handle
);

//*** PermanentCapGetOneHandle()
// This function returns whether a permanent handle exists.
BOOL PermanentCapGetOneHandle(TPM_HANDLE handle  // IN: handle
);

//*** PermanentHandleGetPolicy()
// This function returns a list of the permanent handles of PCR, started from
// 'handle'. If 'handle' is larger than the largest permanent handle, an empty list
// will be returned with 'more' set to NO.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
PermanentHandleGetPolicy(TPM_HANDLE handle,  // IN: start handle
                         UINT32     count,   // IN: max count of returned handles
                         TPML_TAGGED_POLICY* policyList  // OUT: list of handle
);

//*** PermanentHandleGetOnePolicy()
// This function returns a permanent handle's policy, if present.
```

```
BOOL PermanentHandleGetOnePolicy(TPM_HANDLE          handle,  // IN: handle
                                 TPMS_TAGGED_POLICY* policy   // OUT: tagged policy
);

#endif  // _HANDLE_FP_H_
```

6.113 /tpm/include/private/prototypes/HashSequenceStart_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_HashSequenceStart  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_HASHSEQUENCESTART_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_HASHSEQUENCESTART_FP_H_

// Input structure definition
typedef struct
{
    TPM2B_AUTH    auth;
    TPMI_ALG_HASH hashAlg;
} HashSequenceStart_In;

// Output structure definition
typedef struct
{
    TPMI_DH_OBJECT sequenceHandle;
} HashSequenceStart_Out;

// Response code modifiers
#    define RC_HashSequenceStart_auth    (TPM_RC_P + TPM_RC_1)
#    define RC_HashSequenceStart_hashAlg (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_HashSequenceStart(HashSequenceStart_In* in, HashSequenceStart_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_HASHSEQUENCESTART_FP_H_
#endif    // CC_HashSequenceStart
```

6.114 /tpm/include/private/prototypes/Hash_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Hash  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_HASH_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_HASH_FP_H_

// Input structure definition
typedef struct
{
    TPM2B_MAX_BUFFER  data;
    TPMI_ALG_HASH     hashAlg;
    TPMI_RH_HIERARCHY hierarchy;
} Hash_In;

// Output structure definition
typedef struct
{
    TPM2B_DIGEST      outHash;
    TPMT_TK_HASHCHECK validation;
} Hash_Out;

// Response code modifiers
```

```
#    define RC_Hash_data      (TPM_RC_P + TPM_RC_1)
#    define RC_Hash_hashAlg   (TPM_RC_P + TPM_RC_2)
#    define RC_Hash_hierarchy (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_Hash(Hash_In* in, Hash_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_HASH_FP_H_
#endif    // CC_Hash
```

6.115 /tpm/include/private/prototypes/HierarchyChangeAuth_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_HierarchyChangeAuth  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_HIERARCHYCHANGEAUTH_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_HIERARCHYCHANGEAUTH_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_HIERARCHY_AUTH authHandle;
    TPM2B_AUTH             newAuth;
} HierarchyChangeAuth_In;

// Response code modifiers
#    define RC_HierarchyChangeAuth_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_HierarchyChangeAuth_newAuth    (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_HierarchyChangeAuth(HierarchyChangeAuth_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_HIERARCHYCHANGEAUTH_FP_H_
#endif    // CC_HierarchyChangeAuth
```

6.116 /tpm/include/private/prototypes/HierarchyControl_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_HierarchyControl  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_HIERARCHYCONTROL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_HIERARCHYCONTROL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_BASE_HIERARCHY authHandle;
    TPMI_RH_ENABLES        enable;
    TPMI_YES_NO            state;
} HierarchyControl_In;

// Response code modifiers
#    define RC_HierarchyControl_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_HierarchyControl_enable     (TPM_RC_P + TPM_RC_1)
#    define RC_HierarchyControl_state      (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_HierarchyControl(HierarchyControl_In* in);
```

```
#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_HIERARCHYCONTROL_FP_H_
#endif    // CC_HierarchyControl
```

6.117 /tpm/include/private/prototypes/Hierarchy_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 04:23:27PM
 */

#ifndef _HIERARCHY_FP_H_
#define _HIERARCHY_FP_H_

//*** HierarchyPreInstall()
// This function performs the initialization functions for the hierarchy
// when the TPM is simulated. This function should not be called if the
// TPM is not in a manufacturing mode at the manufacturer, or in a simulated
// environment.
void HierarchyPreInstall_Init(void);

//*** HierarchyStartup()
// This function is called at TPM2_Startup() to initialize the hierarchy
// related values.
BOOL HierarchyStartup(STARTUP_TYPE type  // IN: start up type
);

//*** HierarchyGetProof()
// This function derives the proof value associated with a hierarchy. It returns a
// buffer containing the proof value.
//
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM failed
//                              to derive the Firmware SVN Secret for the requested
//                              SVN.
TPM_RC HierarchyGetProof(TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy constant
                         TPM2B_PROOF*      proof       // OUT: proof buffer
);

//*** HierarchyGetPrimarySeed()
// This function derives the primary seed of a hierarchy.
//
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM failed
//                              to derive the Firmware SVN Secret for the requested
//                              SVN.
TPM_RC HierarchyGetPrimarySeed(TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy
                               TPM2B_SEED*       seed        // OUT: seed buffer
);

//*** ValidateHierarchy()
// This function ensures a given hierarchy is valid and enabled.
//  Return Type: TPM_RC
//      TPM_RC_HIERARCHY        Hierarchy is disabled
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the given SVN
```

```
//                              is greater than the TPM's current SVN.
//      TPM_RC_VALUE            Hierarchy is not valid
TPM_RC ValidateHierarchy(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy
);

//*** HierarchyIsEnabled()
// This function checks to see if a hierarchy is enabled.
// NOTE: The TPM_RH_NULL hierarchy is always enabled.
//  Return Type: BOOL
//      TRUE(1)         hierarchy is enabled
//      FALSE(0)        hierarchy is disabled
BOOL HierarchyIsEnabled(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy
);

//*** HierarchyNormalizeHandle
// This function accepts a handle that may or may not be FW- or SVN-bound,
// and returns the base hierarchy to which the handle refers.
TPMI_RH_HIERARCHY HierarchyNormalizeHandle(TPMI_RH_HIERARCHY handle  // IN
);

//*** HierarchyIsFirmwareLimited
// This function accepts a hierarchy handle and returns whether it is firmware-
// limited.
BOOL HierarchyIsFirmwareLimited(TPMI_RH_HIERARCHY handle  // IN
);

//*** HierarchyIsSvnLimited
// This function accepts a hierarchy handle and returns whether it is SVN-
// limited.
BOOL HierarchyIsSvnLimited(TPMI_RH_HIERARCHY handle  // IN
);

#endif  // _HIERARCHY_FP_H_
```

6.118 /tpm/include/private/prototypes/HMAC_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_HMAC  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_HMAC_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_HMAC_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   handle;
    TPM2B_MAX_BUFFER buffer;
    TPMI_ALG_HASH    hashAlg;
} HMAC_In;

// Output structure definition
typedef struct
{
    TPM2B_DIGEST outHMAC;
} HMAC_Out;

// Response code modifiers
#    define RC_HMAC_handle  (TPM_RC_H + TPM_RC_1)
#    define RC_HMAC_buffer  (TPM_RC_P + TPM_RC_1)
#    define RC_HMAC_hashAlg (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_HMAC(HMAC_In* in, HMAC_Out* out);
```

```
#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_HMAC_FP_H_
#endif    // CC_HMAC
```

6.119 /tpm/include/private/prototypes/HMAC_Start_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_HMAC_Start  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_HMAC_START_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_HMAC_START_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT handle;
    TPM2B_AUTH     auth;
    TPMI_ALG_HASH  hashAlg;
} HMAC_Start_In;

// Output structure definition
typedef struct
{
    TPMI_DH_OBJECT sequenceHandle;
} HMAC_Start_Out;

// Response code modifiers
#    define RC_HMAC_Start_handle  (TPM_RC_H + TPM_RC_1)
#    define RC_HMAC_Start_auth    (TPM_RC_P + TPM_RC_1)
#    define RC_HMAC_Start_hashAlg (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_HMAC_Start(HMAC_Start_In* in, HMAC_Start_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_HMAC_START_FP_H_
#endif    // CC_HMAC_Start
```

6.120 /tpm/include/private/prototypes/Import_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Import  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_IMPORT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_IMPORT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT          parentHandle;
    TPM2B_DATA              encryptionKey;
    TPM2B_PUBLIC            objectPublic;
    TPM2B_PRIVATE           duplicate;
    TPM2B_ENCRYPTED_SECRET inSymSeed;
    TPMT_SYM_DEF_OBJECT     symmetricAlg;
} Import_In;

// Output structure definition
typedef struct
{
    TPM2B_PRIVATE outPrivate;
} Import_Out;
```

```
// Response code modifiers
#    define RC_Import_parentHandle  (TPM_RC_H + TPM_RC_1)
#    define RC_Import_encryptionKey (TPM_RC_P + TPM_RC_1)
#    define RC_Import_objectPublic  (TPM_RC_P + TPM_RC_2)
#    define RC_Import_duplicate     (TPM_RC_P + TPM_RC_3)
#    define RC_Import_inSymSeed     (TPM_RC_P + TPM_RC_4)
#    define RC_Import_symmetricAlg  (TPM_RC_P + TPM_RC_5)

// Function prototype
TPM_RC
TPM2_Import(Import_In* in, Import_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_IMPORT_FP_H_
#endif    // CC_Import
```

6.121 /tpm/include/private/prototypes/IncrementalSelfTest_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_IncrementalSelfTest  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_INCREMENTALSELFTEST_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_INCREMENTALSELFTEST_FP_H_

// Input structure definition
typedef struct
{
    TPML_ALG toTest;
} IncrementalSelfTest_In;

// Output structure definition
typedef struct
{
    TPML_ALG toDoList;
} IncrementalSelfTest_Out;

// Response code modifiers
#    define RC_IncrementalSelfTest_toTest (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_IncrementalSelfTest(IncrementalSelfTest_In* in, IncrementalSelfTest_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_INCREMENTALSELFTEST_FP_H_
#endif    // CC_IncrementalSelfTest
```

6.122 /tpm/include/private/prototypes/IoBuffers_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _IO_BUFFERS_FP_H_
#define _IO_BUFFERS_FP_H_

//*** MemoryIoBufferAllocationReset()
// This function is used to reset the allocation of buffers.
void MemoryIoBufferAllocationReset(void);

//*** MemoryIoBufferZero()
// Function zeros the action I/O buffer at the end of a command. Calling this is
// not mandatory for proper functionality.
```

```
void MemoryIoBufferZero(void);

//*** MemoryGetInBuffer()
// This function returns the address of the buffer into which the
// command parameters will be unmarshaled in preparation for calling
// the command actions.
BYTE* MemoryGetInBuffer(UINT32 size  // Size, in bytes, required for the input
                                     // unmarshaling
);

//*** MemoryGetOutBuffer()
// This function returns the address of the buffer into which the command
// action code places its output values.
BYTE* MemoryGetOutBuffer(UINT32 size  // required size of the buffer
);

//*** IsLabelProperlyFormatted()
// This function checks that a label is a null-terminated string.
// NOTE: this function is here because there was no better place for it.
//  Return Type: BOOL
//      TRUE(1)         string is null terminated
//      FALSE(0)        string is not null terminated
BOOL IsLabelProperlyFormatted(TPM2B* x);

#endif  // _IO_BUFFERS_FP_H_
```

6.123 /tpm/include/private/prototypes/LoadExternal_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_LoadExternal  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_LOADEXTERNAL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_LOADEXTERNAL_FP_H_

// Input structure definition
typedef struct
{
    TPM2B_SENSITIVE   inPrivate;
    TPM2B_PUBLIC      inPublic;
    TPMI_RH_HIERARCHY hierarchy;
} LoadExternal_In;

// Output structure definition
typedef struct
{
    TPM_HANDLE objectHandle;
    TPM2B_NAME name;
} LoadExternal_Out;

// Response code modifiers
#    define RC_LoadExternal_inPrivate (TPM_RC_P + TPM_RC_1)
#    define RC_LoadExternal_inPublic  (TPM_RC_P + TPM_RC_2)
#    define RC_LoadExternal_hierarchy (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_LoadExternal(LoadExternal_In* in, LoadExternal_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_LOADEXTERNAL_FP_H_
#endif    // CC_LoadExternal
```

6.124 /tpm/include/private/prototypes/Load_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Load  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_LOAD_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_LOAD_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT parentHandle;
    TPM2B_PRIVATE  inPrivate;
    TPM2B_PUBLIC   inPublic;
} Load_In;

// Output structure definition
typedef struct
{
    TPM_HANDLE objectHandle;
    TPM2B_NAME name;
} Load_Out;

// Response code modifiers
#    define RC_Load_parentHandle (TPM_RC_H + TPM_RC_1)
#    define RC_Load_inPrivate    (TPM_RC_P + TPM_RC_1)
#    define RC_Load_inPublic     (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_Load(Load_In* in, Load_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_LOAD_FP_H_
#endif    // CC_Load
```

6.125 /tpm/include/private/prototypes/Locality_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _LOCALITY_FP_H_
#define _LOCALITY_FP_H_

//** LocalityGetAttributes()
// This function will convert a locality expressed as an integer into
// TPMA_LOCALITY form.
//
// The function returns the locality attribute.
TPMA_LOCALITY
LocalityGetAttributes(UINT8 locality  // IN: locality value
);

#endif  // _LOCALITY_FP_H_
```

6.126 /tpm/include/private/prototypes/MAC_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_MAC  // Command must be enabled
```

```
#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAC_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAC_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT      handle;
    TPM2B_MAX_BUFFER    buffer;
    TPMI_ALG_MAC_SCHEME inScheme;
} MAC_In;

// Output structure definition
typedef struct
{
    TPM2B_DIGEST outMAC;
} MAC_Out;

// Response code modifiers
#    define RC_MAC_handle   (TPM_RC_H + TPM_RC_1)
#    define RC_MAC_buffer   (TPM_RC_P + TPM_RC_1)
#    define RC_MAC_inScheme (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_MAC(MAC_In* in, MAC_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAC_FP_H_
#endif    // CC_MAC
```

6.127 /tpm/include/private/prototypes/MAC_Start_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_MAC_Start  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAC_START_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAC_START_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT      handle;
    TPM2B_AUTH          auth;
    TPMI_ALG_MAC_SCHEME inScheme;
} MAC_Start_In;

// Output structure definition
typedef struct
{
    TPMI_DH_OBJECT sequenceHandle;
} MAC_Start_Out;

// Response code modifiers
#    define RC_MAC_Start_handle   (TPM_RC_H + TPM_RC_1)
#    define RC_MAC_Start_auth     (TPM_RC_P + TPM_RC_1)
#    define RC_MAC_Start_inScheme (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_MAC_Start(MAC_Start_In* in, MAC_Start_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAC_START_FP_H_
#endif    // CC_MAC_Start
```

6.128 /tpm/include/private/prototypes/MakeCredential_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_MakeCredential  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAKECREDENTIAL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAKECREDENTIAL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT handle;
    TPM2B_DIGEST   credential;
    TPM2B_NAME     objectName;
} MakeCredential_In;

// Output structure definition
typedef struct
{
    TPM2B_ID_OBJECT        credentialBlob;
    TPM2B_ENCRYPTED_SECRET secret;
} MakeCredential_Out;

// Response code modifiers
#    define RC_MakeCredential_handle     (TPM_RC_H + TPM_RC_1)
#    define RC_MakeCredential_credential (TPM_RC_P + TPM_RC_1)
#    define RC_MakeCredential_objectName (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_MakeCredential(MakeCredential_In* in, MakeCredential_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_MAKECREDENTIAL_FP_H_
#endif    // CC_MakeCredential
```

6.129 /tpm/include/private/prototypes/Marshal_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#ifndef _MARSHAL_FP_H_
#define _MARSHAL_FP_H_

// Table "Definition of Base Types" (Part 2: Structures)
//   UINT8 definition
TPM_RC
UINT8_Unmarshal(UINT8* target, BYTE** buffer, INT32* size);
UINT16
UINT8_Marshal(UINT8* source, BYTE** buffer, INT32* size);

//   BYTE definition
#if !USE_MARSHALING_DEFINES
TPM_RC
BYTE_Unmarshal(BYTE* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define BYTE_Unmarshal(target, buffer, size) \
      UINT8_Unmarshal((UINT8*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
BYTE_Marshal(BYTE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define BYTE_Marshal(source, buffer, size) \
      UINT8_Marshal((UINT8*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
```

```
//   INT8 definition
#if !USE_MARSHALING_DEFINES
TPM_RC
INT8_Unmarshal(INT8* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT8_Unmarshal(target, buffer, size) \
      UINT8_Unmarshal((UINT8*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
INT8_Marshal(INT8* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT8_Marshal(source, buffer, size) \
      UINT8_Marshal((UINT8*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

//   UINT16 definition
TPM_RC
UINT16_Unmarshal(UINT16* target, BYTE** buffer, INT32* size);
UINT16
UINT16_Marshal(UINT16* source, BYTE** buffer, INT32* size);

//   INT16 definition
#if !USE_MARSHALING_DEFINES
TPM_RC
INT16_Unmarshal(INT16* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT16_Unmarshal(target, buffer, size) \
      UINT16_Unmarshal((UINT16*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
INT16_Marshal(INT16* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT16_Marshal(source, buffer, size) \
      UINT16_Marshal((UINT16*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

//   UINT32 definition
TPM_RC
UINT32_Unmarshal(UINT32* target, BYTE** buffer, INT32* size);
UINT16
UINT32_Marshal(UINT32* source, BYTE** buffer, INT32* size);

//   INT32 definition
#if !USE_MARSHALING_DEFINES
TPM_RC
INT32_Unmarshal(INT32* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT32_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
INT32_Marshal(INT32* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT32_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

//   UINT64 definition
TPM_RC
UINT64_Unmarshal(UINT64* target, BYTE** buffer, INT32* size);
UINT16
UINT64_Marshal(UINT64* source, BYTE** buffer, INT32* size);
```

```
//   INT64 definition
#if !USE_MARSHALING_DEFINES
TPM_RC
INT64_Unmarshal(INT64* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT64_Unmarshal(target, buffer, size) \
      UINT64_Unmarshal((UINT64*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
INT64_Marshal(INT64* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define INT64_Marshal(source, buffer, size) \
      UINT64_Marshal((UINT64*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for Documentation Clarity" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_ALGORITHM_ID_Unmarshal(TPM_ALGORITHM_ID* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_ALGORITHM_ID_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_ALGORITHM_ID_Marshal(TPM_ALGORITHM_ID* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_ALGORITHM_ID_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_AUTHORIZATION_SIZE_Unmarshal(
    TPM_AUTHORIZATION_SIZE* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_AUTHORIZATION_SIZE_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_AUTHORIZATION_SIZE_Marshal(
    TPM_AUTHORIZATION_SIZE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_AUTHORIZATION_SIZE_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_KEY_BITS_Unmarshal(TPM_KEY_BITS* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_KEY_BITS_Unmarshal(target, buffer, size) \
      UINT16_Unmarshal((UINT16*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_KEY_BITS_Marshal(TPM_KEY_BITS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_KEY_BITS_Marshal(source, buffer, size) \
      UINT16_Marshal((UINT16*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_KEY_SIZE_Unmarshal(TPM_KEY_SIZE* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
```

```
#  define TPM_KEY_SIZE_Unmarshal(target, buffer, size) \
      UINT16_Unmarshal((UINT16*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_KEY_SIZE_Marshal(TPM_KEY_SIZE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_KEY_SIZE_Marshal(source, buffer, size) \
      UINT16_Marshal((UINT16*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_MODIFIER_INDICATOR_Unmarshal(
    TPM_MODIFIER_INDICATOR* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_MODIFIER_INDICATOR_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_MODIFIER_INDICATOR_Marshal(
    TPM_MODIFIER_INDICATOR* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_MODIFIER_INDICATOR_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_PARAMETER_SIZE_Unmarshal(TPM_PARAMETER_SIZE* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_PARAMETER_SIZE_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_PARAMETER_SIZE_Marshal(TPM_PARAMETER_SIZE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_PARAMETER_SIZE_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_CONSTANTS32 Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPM_CONSTANTS32_Marshal(TPM_CONSTANTS32* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_CONSTANTS32_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_ALG_ID Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_ALG_ID_Unmarshal(TPM_ALG_ID* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_ALG_ID_Unmarshal(target, buffer, size) \
      UINT16_Unmarshal((UINT16*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_ALG_ID_Marshal(TPM_ALG_ID* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_ALG_ID_Marshal(source, buffer, size) \
      UINT16_Marshal((UINT16*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of TPM_ECC_CURVE Constants" (Part 2: Structures)
TPM_RC
TPM_ECC_CURVE_Unmarshal(TPM_ECC_CURVE* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPM_ECC_CURVE_Marshal(TPM_ECC_CURVE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_ECC_CURVE_Marshal(source, buffer, size) \
      UINT16_Marshal((UINT16*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_CC Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_CC_Unmarshal(TPM_CC* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_CC_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_CC_Marshal(TPM_CC* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_CC_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_RC Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPM_RC_Marshal(TPM_RC* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_RC_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_CLOCK_ADJUST Constants" (Part 2: Structures)
TPM_RC
TPM_CLOCK_ADJUST_Unmarshal(TPM_CLOCK_ADJUST* target, BYTE** buffer, INT32* size);

// Table "Definition of TPM_EO Constants" (Part 2: Structures)
TPM_RC
TPM_EO_Unmarshal(TPM_EO* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPM_EO_Marshal(TPM_EO* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_EO_Marshal(source, buffer, size) \
      UINT16_Marshal((UINT16*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_ST Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_ST_Unmarshal(TPM_ST* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_ST_Unmarshal(target, buffer, size) \
      UINT16_Unmarshal((UINT16*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_ST_Marshal(TPM_ST* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_ST_Marshal(source, buffer, size) \
      UINT16_Marshal((UINT16*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of TPM_SU Constants" (Part 2: Structures)
TPM_RC
TPM_SU_Unmarshal(TPM_SU* target, BYTE** buffer, INT32* size);

// Table "Definition of TPM_SE Constants" (Part 2: Structures)
TPM_RC
TPM_SE_Unmarshal(TPM_SE* target, BYTE** buffer, INT32* size);

// Table "Definition of TPM_CAP Constants" (Part 2: Structures)
TPM_RC
TPM_CAP_Unmarshal(TPM_CAP* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPM_CAP_Marshal(TPM_CAP* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_CAP_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_PT Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_PT_Unmarshal(TPM_PT* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_PT_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_PT_Marshal(TPM_PT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_PT_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_PT_PCR Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_PT_PCR_Unmarshal(TPM_PT_PCR* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_PT_PCR_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_PT_PCR_Marshal(TPM_PT_PCR* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_PT_PCR_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_PS Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPM_PS_Marshal(TPM_PS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_PS_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for Handles" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_HANDLE_Unmarshal(TPM_HANDLE* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
```

```
#  define TPM_HANDLE_Unmarshal(target, buffer, size) \
      UINT32_Unmarshal((UINT32*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_HANDLE_Marshal(TPM_HANDLE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_HANDLE_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_HT Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_HT_Unmarshal(TPM_HT* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_HT_Unmarshal(target, buffer, size) \
      UINT8_Unmarshal((UINT8*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_HT_Marshal(TPM_HT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_HT_Marshal(source, buffer, size) \
      UINT8_Marshal((UINT8*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_RH Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_RH_Unmarshal(TPM_RH* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_RH_Unmarshal(target, buffer, size) \
      TPM_HANDLE_Unmarshal((TPM_HANDLE*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_RH_Marshal(TPM_RH* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_RH_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_HC Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM_HC_Unmarshal(TPM_HC* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_HC_Unmarshal(target, buffer, size) \
      TPM_HANDLE_Unmarshal((TPM_HANDLE*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM_HC_Marshal(TPM_HC* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_HC_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_ALGORITHM Bits" (Part 2: Structures)
TPM_RC
TPMA_ALGORITHM_Unmarshal(TPMA_ALGORITHM* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_ALGORITHM_Marshal(TPMA_ALGORITHM* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
```

```
#  define TPMA_ALGORITHM_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_OBJECT Bits" (Part 2: Structures)
TPM_RC
TPMA_OBJECT_Unmarshal(TPMA_OBJECT* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_OBJECT_Marshal(TPMA_OBJECT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_OBJECT_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_SESSION Bits" (Part 2: Structures)
TPM_RC
TPMA_SESSION_Unmarshal(TPMA_SESSION* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_SESSION_Marshal(TPMA_SESSION* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_SESSION_Marshal(source, buffer, size) \
      UINT8_Marshal((UINT8*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_LOCALITY Bits" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMA_LOCALITY_Unmarshal(TPMA_LOCALITY* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_LOCALITY_Unmarshal(target, buffer, size) \
      UINT8_Unmarshal((UINT8*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_LOCALITY_Marshal(TPMA_LOCALITY* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_LOCALITY_Marshal(source, buffer, size) \
      UINT8_Marshal((UINT8*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_PERMANENT Bits" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_PERMANENT_Marshal(TPMA_PERMANENT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_PERMANENT_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_STARTUP_CLEAR Bits" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_STARTUP_CLEAR_Marshal(TPMA_STARTUP_CLEAR* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_STARTUP_CLEAR_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_MEMORY Bits" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_MEMORY_Marshal(TPMA_MEMORY* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_MEMORY_Marshal(source, buffer, size) \
```

```
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_CC Bits" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_CC_Marshal(TPMA_CC* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_CC_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_MODES Bits" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_MODES_Marshal(TPMA_MODES* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_MODES_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_ACT Bits" (Part 2: Structures)
TPM_RC
TPMA_ACT_Unmarshal(TPMA_ACT* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_ACT_Marshal(TPMA_ACT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_ACT_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_YES_NO Type" (Part 2: Structures)
TPM_RC
TPMI_YES_NO_Unmarshal(TPMI_YES_NO* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_YES_NO_Marshal(TPMI_YES_NO* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_YES_NO_Marshal(source, buffer, size) \
      BYTE_Marshal((BYTE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_OBJECT Type" (Part 2: Structures)
TPM_RC
TPMI_DH_OBJECT_Unmarshal(
    TPMI_DH_OBJECT* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_OBJECT_Marshal(TPMI_DH_OBJECT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_DH_OBJECT_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_PARENT Type" (Part 2: Structures)
TPM_RC
TPMI_DH_PARENT_Unmarshal(TPMI_DH_PARENT* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_PARENT_Marshal(TPMI_DH_PARENT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_DH_PARENT_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of TPMI_DH_PERSISTENT Type" (Part 2: Structures)
TPM_RC
TPMI_DH_PERSISTENT_Unmarshal(TPMI_DH_PERSISTENT* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_PERSISTENT_Marshal(TPMI_DH_PERSISTENT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_DH_PERSISTENT_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_ENTITY Type" (Part 2: Structures)
TPM_RC
TPMI_DH_ENTITY_Unmarshal(
    TPMI_DH_ENTITY* target, BYTE** buffer, INT32* size, BOOL flag);

// Table "Definition of TPMI_DH_PCR Type" (Part 2: Structures)
TPM_RC
TPMI_DH_PCR_Unmarshal(TPMI_DH_PCR* target, BYTE** buffer, INT32* size, BOOL flag);

// Table "Definition of TPMI_SH_AUTH_SESSION Type" (Part 2: Structures)
TPM_RC
TPMI_SH_AUTH_SESSION_Unmarshal(
    TPMI_SH_AUTH_SESSION* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_SH_AUTH_SESSION_Marshal(
    TPMI_SH_AUTH_SESSION* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_SH_AUTH_SESSION_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_SH_HMAC Type" (Part 2: Structures)
TPM_RC
TPMI_SH_HMAC_Unmarshal(TPMI_SH_HMAC* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_SH_HMAC_Marshal(TPMI_SH_HMAC* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_SH_HMAC_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_SH_POLICY Type" (Part 2: Structures)
TPM_RC
TPMI_SH_POLICY_Unmarshal(TPMI_SH_POLICY* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_SH_POLICY_Marshal(TPMI_SH_POLICY* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_SH_POLICY_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_CONTEXT Type" (Part 2: Structures)
TPM_RC
TPMI_DH_CONTEXT_Unmarshal(TPMI_DH_CONTEXT* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_CONTEXT_Marshal(TPMI_DH_CONTEXT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_DH_CONTEXT_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of TPMI_DH_SAVED Type" (Part 2: Structures)
TPM_RC
TPMI_DH_SAVED_Unmarshal(TPMI_DH_SAVED* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_SAVED_Marshal(TPMI_DH_SAVED* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_DH_SAVED_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_HIERARCHY Type" (Part 2: Structures)
TPM_RC
TPMI_RH_HIERARCHY_Unmarshal(TPMI_RH_HIERARCHY* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_HIERARCHY_Marshal(TPMI_RH_HIERARCHY* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RH_HIERARCHY_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_ENABLES Type" (Part 2: Structures)
TPM_RC
TPMI_RH_ENABLES_Unmarshal(
    TPMI_RH_ENABLES* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_ENABLES_Marshal(TPMI_RH_ENABLES* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RH_ENABLES_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_HIERARCHY_AUTH Type" (Part 2: Structures)
TPM_RC
TPMI_RH_HIERARCHY_AUTH_Unmarshal(
    TPMI_RH_HIERARCHY_AUTH* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_HIERARCHY_POLICY Type" (Part 2: Structures)
TPM_RC
TPMI_RH_HIERARCHY_POLICY_Unmarshal(
    TPMI_RH_HIERARCHY_POLICY* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_BASE_HIERARCHY Type" (Part 2: Structures)
TPM_RC
TPMI_RH_BASE_HIERARCHY_Unmarshal(
    TPMI_RH_BASE_HIERARCHY* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_BASE_HIERARCHY_Marshal(
    TPMI_RH_BASE_HIERARCHY* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RH_BASE_HIERARCHY_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_PLATFORM Type" (Part 2: Structures)
TPM_RC
TPMI_RH_PLATFORM_Unmarshal(TPMI_RH_PLATFORM* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_OWNER Type" (Part 2: Structures)
TPM_RC
TPMI_RH_OWNER_Unmarshal(TPMI_RH_OWNER* target, BYTE** buffer, INT32* size, BOOL flag);

// Table "Definition of TPMI_RH_ENDORSEMENT Type" (Part 2: Structures)
```

```
TPM_RC
TPMI_RH_ENDORSEMENT_Unmarshal(
    TPMI_RH_ENDORSEMENT* target, BYTE** buffer, INT32* size, BOOL flag);

// Table "Definition of TPMI_RH_PROVISION Type" (Part 2: Structures)
TPM_RC
TPMI_RH_PROVISION_Unmarshal(TPMI_RH_PROVISION* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_CLEAR Type" (Part 2: Structures)
TPM_RC
TPMI_RH_CLEAR_Unmarshal(TPMI_RH_CLEAR* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_NV_AUTH Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_AUTH_Unmarshal(TPMI_RH_NV_AUTH* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_LOCKOUT Type" (Part 2: Structures)
TPM_RC
TPMI_RH_LOCKOUT_Unmarshal(TPMI_RH_LOCKOUT* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_NV_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_INDEX_Unmarshal(TPMI_RH_NV_INDEX* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_NV_INDEX_Marshal(TPMI_RH_NV_INDEX* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RH_NV_INDEX_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_NV_DEFINED_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_DEFINED_INDEX_Unmarshal(
    TPMI_RH_NV_DEFINED_INDEX* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RH_NV_LEGACY_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_LEGACY_INDEX_Unmarshal(
    TPMI_RH_NV_LEGACY_INDEX* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_NV_LEGACY_INDEX_Marshal(
    TPMI_RH_NV_LEGACY_INDEX* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RH_NV_LEGACY_INDEX_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_NV_EXP_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_EXP_INDEX_Unmarshal(
    TPMI_RH_NV_EXP_INDEX* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_NV_EXP_INDEX_Marshal(
    TPMI_RH_NV_EXP_INDEX* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RH_NV_EXP_INDEX_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_AC Type" (Part 2: Structures)
TPM_RC
TPMI_RH_AC_Unmarshal(TPMI_RH_AC* target, BYTE** buffer, INT32* size);
```

```
// Table "Definition of TPMI_RH_ACT Type" (Part 2: Structures)
TPM_RC
TPMI_RH_ACT_Unmarshal(TPMI_RH_ACT* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_ACT_Marshal(TPMI_RH_ACT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RH_ACT_Marshal(source, buffer, size) \
      TPM_HANDLE_Marshal((TPM_HANDLE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_HASH Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_HASH_Unmarshal(TPMI_ALG_HASH* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_HASH_Marshal(TPMI_ALG_HASH* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_HASH_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_ASYM Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_ASYM_Unmarshal(TPMI_ALG_ASYM* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_ASYM_Marshal(TPMI_ALG_ASYM* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_ASYM_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SYM Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SYM_Unmarshal(TPMI_ALG_SYM* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SYM_Marshal(TPMI_ALG_SYM* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_SYM_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SYM_OBJECT Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SYM_OBJECT_Unmarshal(
    TPMI_ALG_SYM_OBJECT* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SYM_OBJECT_Marshal(TPMI_ALG_SYM_OBJECT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_SYM_OBJECT_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SYM_MODE Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SYM_MODE_Unmarshal(
    TPMI_ALG_SYM_MODE* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SYM_MODE_Marshal(TPMI_ALG_SYM_MODE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_SYM_MODE_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
```

```
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_KDF Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_KDF_Unmarshal(TPMI_ALG_KDF* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_KDF_Marshal(TPMI_ALG_KDF* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_KDF_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SIG_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SIG_SCHEME_Unmarshal(
    TPMI_ALG_SIG_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SIG_SCHEME_Marshal(TPMI_ALG_SIG_SCHEME* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_SIG_SCHEME_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ECC_KEY_EXCHANGE Type" (Part 2: Structures)
TPM_RC
TPMI_ECC_KEY_EXCHANGE_Unmarshal(
    TPMI_ECC_KEY_EXCHANGE* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ECC_KEY_EXCHANGE_Marshal(
    TPMI_ECC_KEY_EXCHANGE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ECC_KEY_EXCHANGE_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ST_COMMAND_TAG Type" (Part 2: Structures)
TPM_RC
TPMI_ST_COMMAND_TAG_Unmarshal(
    TPMI_ST_COMMAND_TAG* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ST_COMMAND_TAG_Marshal(TPMI_ST_COMMAND_TAG* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ST_COMMAND_TAG_Marshal(source, buffer, size) \
      TPM_ST_Marshal((TPM_ST*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_MAC_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_MAC_SCHEME_Unmarshal(
    TPMI_ALG_MAC_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_MAC_SCHEME_Marshal(TPMI_ALG_MAC_SCHEME* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_MAC_SCHEME_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_CIPHER_MODE Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_CIPHER_MODE_Unmarshal(
    TPMI_ALG_CIPHER_MODE* target, BYTE** buffer, INT32* size, BOOL flag);
```

```
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_CIPHER_MODE_Marshal(
    TPMI_ALG_CIPHER_MODE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_CIPHER_MODE_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_EMPTY Structure" (Part 2: Structures)
TPM_RC
TPMS_EMPTY_Unmarshal(TPMS_EMPTY* target, BYTE** buffer, INT32* size);
UINT16
TPMS_EMPTY_Marshal(TPMS_EMPTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_ALGORITHM_DESCRIPTION Structure" (Part 2: Structures)
UINT16
TPMS_ALGORITHM_DESCRIPTION_Marshal(
    TPMS_ALGORITHM_DESCRIPTION* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMU_HA Union" (Part 2: Structures)
TPM_RC
TPMU_HA_Unmarshal(TPMU_HA* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_HA_Marshal(TPMU_HA* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_HA Structure" (Part 2: Structures)
TPM_RC
TPMT_HA_Unmarshal(TPMT_HA* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_HA_Marshal(TPMT_HA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_DIGEST Structure" (Part 2: Structures)
TPM_RC
TPM2B_DIGEST_Unmarshal(TPM2B_DIGEST* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_DIGEST_Marshal(TPM2B_DIGEST* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_DATA_Unmarshal(TPM2B_DATA* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_DATA_Marshal(TPM2B_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of Types for TPM2B_NONCE" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM2B_NONCE_Unmarshal(TPM2B_NONCE* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM2B_NONCE_Unmarshal(target, buffer, size) \
      TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM2B_NONCE_Marshal(TPM2B_NONCE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM2B_NONCE_Marshal(source, buffer, size) \
      TPM2B_DIGEST_Marshal((TPM2B_DIGEST*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for TPM2B_AUTH" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM2B_AUTH_Unmarshal(TPM2B_AUTH* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM2B_AUTH_Unmarshal(target, buffer, size) \
```

```
      TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM2B_AUTH_Marshal(TPM2B_AUTH* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM2B_AUTH_Marshal(source, buffer, size) \
      TPM2B_DIGEST_Marshal((TPM2B_DIGEST*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for TPM2B_OPERAND" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPM2B_OPERAND_Unmarshal(TPM2B_OPERAND* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM2B_OPERAND_Unmarshal(target, buffer, size) \
      TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPM2B_OPERAND_Marshal(TPM2B_OPERAND* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM2B_OPERAND_Marshal(source, buffer, size) \
      TPM2B_DIGEST_Marshal((TPM2B_DIGEST*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM2B_EVENT Structure" (Part 2: Structures)
TPM_RC
TPM2B_EVENT_Unmarshal(TPM2B_EVENT* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_EVENT_Marshal(TPM2B_EVENT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_MAX_BUFFER Structure" (Part 2: Structures)
TPM_RC
TPM2B_MAX_BUFFER_Unmarshal(TPM2B_MAX_BUFFER* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_MAX_BUFFER_Marshal(TPM2B_MAX_BUFFER* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_MAX_NV_BUFFER Structure" (Part 2: Structures)
TPM_RC
TPM2B_MAX_NV_BUFFER_Unmarshal(
    TPM2B_MAX_NV_BUFFER* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_MAX_NV_BUFFER_Marshal(TPM2B_MAX_NV_BUFFER* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_TIMEOUT Structure" (Part 2: Structures)
TPM_RC
TPM2B_TIMEOUT_Unmarshal(TPM2B_TIMEOUT* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_TIMEOUT_Marshal(TPM2B_TIMEOUT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_IV Structure" (Part 2: Structures)
TPM_RC
TPM2B_IV_Unmarshal(TPM2B_IV* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_IV_Marshal(TPM2B_IV* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_VENDOR_PROPERTY Structure" (Part 2: Structures)
TPM_RC
TPM2B_VENDOR_PROPERTY_Unmarshal(
    TPM2B_VENDOR_PROPERTY* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_VENDOR_PROPERTY_Marshal(
    TPM2B_VENDOR_PROPERTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_NAME Structure" (Part 2: Structures)
```

```
TPM_RC
TPM2B_NAME_Unmarshal(TPM2B_NAME* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_NAME_Marshal(TPM2B_NAME* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_PCR_SELECT Structure" (Part 2: Structures)
TPM_RC
TPMS_PCR_SELECT_Unmarshal(TPMS_PCR_SELECT* target, BYTE** buffer, INT32* size);
UINT16
TPMS_PCR_SELECT_Marshal(TPMS_PCR_SELECT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_PCR_SELECTION Structure" (Part 2: Structures)
TPM_RC
TPMS_PCR_SELECTION_Unmarshal(TPMS_PCR_SELECTION* target, BYTE** buffer, INT32* size);
UINT16
TPMS_PCR_SELECTION_Marshal(TPMS_PCR_SELECTION* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMT_TK_CREATION Structure" (Part 2: Structures)
TPM_RC
TPMT_TK_CREATION_Unmarshal(TPMT_TK_CREATION* target, BYTE** buffer, INT32* size);
UINT16
TPMT_TK_CREATION_Marshal(TPMT_TK_CREATION* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMT_TK_VERIFIED Structure" (Part 2: Structures)
TPM_RC
TPMT_TK_VERIFIED_Unmarshal(TPMT_TK_VERIFIED* target, BYTE** buffer, INT32* size);
UINT16
TPMT_TK_VERIFIED_Marshal(TPMT_TK_VERIFIED* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMT_TK_AUTH Structure" (Part 2: Structures)
TPM_RC
TPMT_TK_AUTH_Unmarshal(TPMT_TK_AUTH* target, BYTE** buffer, INT32* size);
UINT16
TPMT_TK_AUTH_Marshal(TPMT_TK_AUTH* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMT_TK_HASHCHECK Structure" (Part 2: Structures)
TPM_RC
TPMT_TK_HASHCHECK_Unmarshal(TPMT_TK_HASHCHECK* target, BYTE** buffer, INT32* size);
UINT16
TPMT_TK_HASHCHECK_Marshal(TPMT_TK_HASHCHECK* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_ALG_PROPERTY Structure" (Part 2: Structures)
UINT16
TPMS_ALG_PROPERTY_Marshal(TPMS_ALG_PROPERTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_TAGGED_PROPERTY Structure" (Part 2: Structures)
UINT16
TPMS_TAGGED_PROPERTY_Marshal(
    TPMS_TAGGED_PROPERTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_TAGGED_PCR_SELECT Structure" (Part 2: Structures)
UINT16
TPMS_TAGGED_PCR_SELECT_Marshal(
    TPMS_TAGGED_PCR_SELECT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_TAGGED_POLICY Structure" (Part 2: Structures)
UINT16
TPMS_TAGGED_POLICY_Marshal(TPMS_TAGGED_POLICY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_ACT_DATA Structure" (Part 2: Structures)
UINT16
TPMS_ACT_DATA_Marshal(TPMS_ACT_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_CC Structure" (Part 2: Structures)
TPM_RC
TPML_CC_Unmarshal(TPML_CC* target, BYTE** buffer, INT32* size);
```

```
UINT16
TPML_CC_Marshal(TPML_CC* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_CCA Structure" (Part 2: Structures)
UINT16
TPML_CCA_Marshal(TPML_CCA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_ALG Structure" (Part 2: Structures)
TPM_RC
TPML_ALG_Unmarshal(TPML_ALG* target, BYTE** buffer, INT32* size);
UINT16
TPML_ALG_Marshal(TPML_ALG* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_HANDLE Structure" (Part 2: Structures)
UINT16
TPML_HANDLE_Marshal(TPML_HANDLE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_DIGEST Structure" (Part 2: Structures)
TPM_RC
TPML_DIGEST_Unmarshal(TPML_DIGEST* target, BYTE** buffer, INT32* size);
UINT16
TPML_DIGEST_Marshal(TPML_DIGEST* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_DIGEST_VALUES Structure" (Part 2: Structures)
TPM_RC
TPML_DIGEST_VALUES_Unmarshal(TPML_DIGEST_VALUES* target, BYTE** buffer, INT32* size);
UINT16
TPML_DIGEST_VALUES_Marshal(TPML_DIGEST_VALUES* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_PCR_SELECTION Structure" (Part 2: Structures)
TPM_RC
TPML_PCR_SELECTION_Unmarshal(TPML_PCR_SELECTION* target, BYTE** buffer, INT32* size);
UINT16
TPML_PCR_SELECTION_Marshal(TPML_PCR_SELECTION* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_ALG_PROPERTY Structure" (Part 2: Structures)
UINT16
TPML_ALG_PROPERTY_Marshal(TPML_ALG_PROPERTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_TAGGED_TPM_PROPERTY Structure" (Part 2: Structures)
UINT16
TPML_TAGGED_TPM_PROPERTY_Marshal(
    TPML_TAGGED_TPM_PROPERTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_TAGGED_PCR_PROPERTY Structure" (Part 2: Structures)
UINT16
TPML_TAGGED_PCR_PROPERTY_Marshal(
    TPML_TAGGED_PCR_PROPERTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_ECC_CURVE Structure" (Part 2: Structures)
#if ALG_ECC
UINT16
TPML_ECC_CURVE_Marshal(TPML_ECC_CURVE* source, BYTE** buffer, INT32* size);
#else  // ALG_ECC
#  define TPML_ECC_CURVE_Marshal UNIMPLEMENTED_Marshal
#endif  // ALG_ECC

// Table "Definition of TPML_TAGGED_POLICY Structure" (Part 2: Structures)
UINT16
TPML_TAGGED_POLICY_Marshal(TPML_TAGGED_POLICY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_ACT_DATA Structure" (Part 2: Structures)
UINT16
TPML_ACT_DATA_Marshal(TPML_ACT_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_VENDOR_PROPERTY Structure" (Part 2: Structures)
```

```
TPM_RC
TPML_VENDOR_PROPERTY_Unmarshal(
    TPML_VENDOR_PROPERTY* target, BYTE** buffer, INT32* size);
UINT16
TPML_VENDOR_PROPERTY_Marshal(
    TPML_VENDOR_PROPERTY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMU_CAPABILITIES Union" (Part 2: Structures)
UINT16
TPMU_CAPABILITIES_Marshal(
    TPMU_CAPABILITIES* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMS_CAPABILITY_DATA Structure" (Part 2: Structures)
UINT16
TPMS_CAPABILITY_DATA_Marshal(
    TPMS_CAPABILITY_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMU_SET_CAPABILITIES Structure" (Part 2: Structures)
TPM_RC
TPMU_SET_CAPABILITIES_Unmarshal(
    TPMU_SET_CAPABILITIES* target, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMS_SET_CAPABILITY_DATA Structure" (Part 2: Structures)
TPM_RC
TPMS_SET_CAPABILITY_DATA_Unmarshal(
    TPMS_SET_CAPABILITY_DATA* target, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_SET_CAPABILITY_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_SET_CAPABILITY_DATA_Unmarshal(
    TPM2B_SET_CAPABILITY_DATA* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_CLOCK_INFO Structure" (Part 2: Structures)
TPM_RC
TPMS_CLOCK_INFO_Unmarshal(TPMS_CLOCK_INFO* target, BYTE** buffer, INT32* size);
UINT16
TPMS_CLOCK_INFO_Marshal(TPMS_CLOCK_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_TIME_INFO Structure" (Part 2: Structures)
TPM_RC
TPMS_TIME_INFO_Unmarshal(TPMS_TIME_INFO* target, BYTE** buffer, INT32* size);
UINT16
TPMS_TIME_INFO_Marshal(TPMS_TIME_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_TIME_ATTEST_INFO Structure" (Part 2: Structures)
UINT16
TPMS_TIME_ATTEST_INFO_Marshal(
    TPMS_TIME_ATTEST_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_CERTIFY_INFO Structure" (Part 2: Structures)
UINT16
TPMS_CERTIFY_INFO_Marshal(TPMS_CERTIFY_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_QUOTE_INFO Structure" (Part 2: Structures)
UINT16
TPMS_QUOTE_INFO_Marshal(TPMS_QUOTE_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_COMMAND_AUDIT_INFO Structure" (Part 2: Structures)
UINT16
TPMS_COMMAND_AUDIT_INFO_Marshal(
    TPMS_COMMAND_AUDIT_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_SESSION_AUDIT_INFO Structure" (Part 2: Structures)
UINT16
TPMS_SESSION_AUDIT_INFO_Marshal(
    TPMS_SESSION_AUDIT_INFO* source, BYTE** buffer, INT32* size);
```

```
// Table "Definition of TPMS_CREATION_INFO Structure" (Part 2: Structures)
UINT16
TPMS_CREATION_INFO_Marshal(TPMS_CREATION_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_NV_CERTIFY_INFO Structure" (Part 2: Structures)
UINT16
TPMS_NV_CERTIFY_INFO_Marshal(
    TPMS_NV_CERTIFY_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_NV_DIGEST_CERTIFY_INFO Structure" (Part 2: Structures)
UINT16
TPMS_NV_DIGEST_CERTIFY_INFO_Marshal(
    TPMS_NV_DIGEST_CERTIFY_INFO* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_ST_ATTEST Type" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ST_ATTEST_Marshal(TPMI_ST_ATTEST* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ST_ATTEST_Marshal(source, buffer, size) \
      TPM_ST_Marshal((TPM_ST*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_ATTEST Union" (Part 2: Structures)
UINT16
TPMU_ATTEST_Marshal(TPMU_ATTEST* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMS_ATTEST Structure" (Part 2: Structures)
UINT16
TPMS_ATTEST_Marshal(TPMS_ATTEST* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_ATTEST Structure" (Part 2: Structures)
UINT16
TPM2B_ATTEST_Marshal(TPM2B_ATTEST* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_AUTH_COMMAND Structure" (Part 2: Structures)
TPM_RC
TPMS_AUTH_COMMAND_Unmarshal(TPMS_AUTH_COMMAND* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_AUTH_RESPONSE Structure" (Part 2: Structures)
UINT16
TPMS_AUTH_RESPONSE_Marshal(TPMS_AUTH_RESPONSE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_AES_KEY_BITS Type" (Part 2: Structures)
TPM_RC
TPMI_AES_KEY_BITS_Unmarshal(TPMI_AES_KEY_BITS* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_AES_KEY_BITS_Marshal(TPMI_AES_KEY_BITS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_AES_KEY_BITS_Marshal(source, buffer, size) \
      TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
// Table "Definition of TPMI_SM4_KEY_BITS Type" (Part 2: Structures)
TPM_RC
TPMI_SM4_KEY_BITS_Unmarshal(TPMI_SM4_KEY_BITS* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_SM4_KEY_BITS_Marshal(TPMI_SM4_KEY_BITS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_SM4_KEY_BITS_Marshal(source, buffer, size) \
      TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_CAMELLIA_KEY_BITS Type" (Part 2: Structures)
```

```
TPM_RC
TPMI_CAMELLIA_KEY_BITS_Unmarshal(
    TPMI_CAMELLIA_KEY_BITS* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_CAMELLIA_KEY_BITS_Marshal(
    TPMI_CAMELLIA_KEY_BITS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_CAMELLIA_KEY_BITS_Marshal(source, buffer, size) \
      TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_SYM_KEY_BITS Union" (Part 2: Structures)
TPM_RC
TPMU_SYM_KEY_BITS_Unmarshal(
    TPMU_SYM_KEY_BITS* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_SYM_KEY_BITS_Marshal(
    TPMU_SYM_KEY_BITS* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMU_SYM_MODE Union" (Part 2: Structures)
TPM_RC
TPMU_SYM_MODE_Unmarshal(
    TPMU_SYM_MODE* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_SYM_MODE_Marshal(
    TPMU_SYM_MODE* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_SYM_DEF Structure" (Part 2: Structures)
TPM_RC
TPMT_SYM_DEF_Unmarshal(TPMT_SYM_DEF* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_SYM_DEF_Marshal(TPMT_SYM_DEF* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMT_SYM_DEF_OBJECT Structure" (Part 2: Structures)
TPM_RC
TPMT_SYM_DEF_OBJECT_Unmarshal(
    TPMT_SYM_DEF_OBJECT* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_SYM_DEF_OBJECT_Marshal(TPMT_SYM_DEF_OBJECT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_SYM_KEY Structure" (Part 2: Structures)
TPM_RC
TPM2B_SYM_KEY_Unmarshal(TPM2B_SYM_KEY* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_SYM_KEY_Marshal(TPM2B_SYM_KEY* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_SYMCIPHER_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_SYMCIPHER_PARMS_Unmarshal(
    TPMS_SYMCIPHER_PARMS* target, BYTE** buffer, INT32* size);
UINT16
TPMS_SYMCIPHER_PARMS_Marshal(
    TPMS_SYMCIPHER_PARMS* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_LABEL Structure" (Part 2: Structures)
TPM_RC
TPM2B_LABEL_Unmarshal(TPM2B_LABEL* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_LABEL_Marshal(TPM2B_LABEL* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_DERIVE Structure" (Part 2: Structures)
TPM_RC
TPMS_DERIVE_Unmarshal(TPMS_DERIVE* target, BYTE** buffer, INT32* size);
UINT16
TPMS_DERIVE_Marshal(TPMS_DERIVE* source, BYTE** buffer, INT32* size);
```

```
// Table "Definition of TPM2B_DERIVE Structure" (Part 2: Structures)
TPM_RC
TPM2B_DERIVE_Unmarshal(TPM2B_DERIVE* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_DERIVE_Marshal(TPM2B_DERIVE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_SENSITIVE_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_SENSITIVE_DATA_Unmarshal(
    TPM2B_SENSITIVE_DATA* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_SENSITIVE_DATA_Marshal(
    TPM2B_SENSITIVE_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_SENSITIVE_CREATE Structure" (Part 2: Structures)
TPM_RC
TPMS_SENSITIVE_CREATE_Unmarshal(
    TPMS_SENSITIVE_CREATE* target, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_SENSITIVE_CREATE Structure" (Part 2: Structures)
TPM_RC
TPM2B_SENSITIVE_CREATE_Unmarshal(
    TPM2B_SENSITIVE_CREATE* target, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_SCHEME_HASH Structure" (Part 2: Structures)
TPM_RC
TPMS_SCHEME_HASH_Unmarshal(TPMS_SCHEME_HASH* target, BYTE** buffer, INT32* size);
UINT16
TPMS_SCHEME_HASH_Marshal(TPMS_SCHEME_HASH* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_SCHEME_ECDAA Structure" (Part 2: Structures)
TPM_RC
TPMS_SCHEME_ECDAA_Unmarshal(TPMS_SCHEME_ECDAA* target, BYTE** buffer, INT32* size);
UINT16
TPMS_SCHEME_ECDAA_Marshal(TPMS_SCHEME_ECDAA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_ALG_KEYEDHASH_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_KEYEDHASH_SCHEME_Unmarshal(
    TPMI_ALG_KEYEDHASH_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_KEYEDHASH_SCHEME_Marshal(
    TPMI_ALG_KEYEDHASH_SCHEME* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_KEYEDHASH_SCHEME_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for HMAC_SIG_SCHEME" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SCHEME_HMAC_Unmarshal(TPMS_SCHEME_HMAC* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SCHEME_HMAC_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SCHEME_HMAC_Marshal(TPMS_SCHEME_HMAC* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SCHEME_HMAC_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of TPMS_SCHEME_XOR Structure" (Part 2: Structures)
TPM_RC
TPMS_SCHEME_XOR_Unmarshal(TPMS_SCHEME_XOR* target, BYTE** buffer, INT32* size);
UINT16
TPMS_SCHEME_XOR_Marshal(TPMS_SCHEME_XOR* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMU_SCHEME_KEYEDHASH Union" (Part 2: Structures)
TPM_RC
TPMU_SCHEME_KEYEDHASH_Unmarshal(
    TPMU_SCHEME_KEYEDHASH* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_SCHEME_KEYEDHASH_Marshal(
    TPMU_SCHEME_KEYEDHASH* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_KEYEDHASH_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_KEYEDHASH_SCHEME_Unmarshal(
    TPMT_KEYEDHASH_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_KEYEDHASH_SCHEME_Marshal(
    TPMT_KEYEDHASH_SCHEME* source, BYTE** buffer, INT32* size);

// Table "Definition of Types for RSA Signature Schemes" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_RSASSA_Unmarshal(
    TPMS_SIG_SCHEME_RSASSA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_RSASSA_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIG_SCHEME_RSASSA_Marshal(
    TPMS_SIG_SCHEME_RSASSA* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_RSASSA_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_RSAPSS_Unmarshal(
    TPMS_SIG_SCHEME_RSAPSS* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_RSAPSS_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIG_SCHEME_RSAPSS_Marshal(
    TPMS_SIG_SCHEME_RSAPSS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_RSAPSS_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for ECC Signature Schemes" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_ECDSA_Unmarshal(
    TPMS_SIG_SCHEME_ECDSA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_ECDSA_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
```

```
UINT16
TPMS_SIG_SCHEME_ECDSA_Marshal(
    TPMS_SIG_SCHEME_ECDSA* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_ECDSA_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_ECDAA_Unmarshal(
    TPMS_SIG_SCHEME_ECDAA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_ECDAA_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_ECDAA_Unmarshal((TPMS_SCHEME_ECDAA*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIG_SCHEME_ECDAA_Marshal(
    TPMS_SIG_SCHEME_ECDAA* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_ECDAA_Marshal(source, buffer, size) \
      TPMS_SCHEME_ECDAA_Marshal((TPMS_SCHEME_ECDAA*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_SM2_Unmarshal(
    TPMS_SIG_SCHEME_SM2* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_SM2_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIG_SCHEME_SM2_Marshal(TPMS_SIG_SCHEME_SM2* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_SM2_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_ECSCHNORR_Unmarshal(
    TPMS_SIG_SCHEME_ECSCHNORR* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_ECSCHNORR_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIG_SCHEME_ECSCHNORR_Marshal(
    TPMS_SIG_SCHEME_ECSCHNORR* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_ECSCHNORR_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_EDDSA_Unmarshal(
    TPMS_SIG_SCHEME_EDDSA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_EDDSA_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIG_SCHEME_EDDSA_Marshal(
    TPMS_SIG_SCHEME_EDDSA* source, BYTE** buffer, INT32* size);
```

```
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_EDDSA_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_EDDSA_PH_Unmarshal(
    TPMS_SIG_SCHEME_EDDSA_PH* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_EDDSA_PH_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIG_SCHEME_EDDSA_PH_Marshal(
    TPMS_SIG_SCHEME_EDDSA_PH* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIG_SCHEME_EDDSA_PH_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_SIG_SCHEME Union" (Part 2: Structures)
TPM_RC
TPMU_SIG_SCHEME_Unmarshal(
    TPMU_SIG_SCHEME* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_SIG_SCHEME_Marshal(
    TPMU_SIG_SCHEME* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_SIG_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_SIG_SCHEME_Unmarshal(
    TPMT_SIG_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_SIG_SCHEME_Marshal(TPMT_SIG_SCHEME* source, BYTE** buffer, INT32* size);

// Table "Definition of Types for Encryption Schemes" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_ENC_SCHEME_RSAES_Unmarshal(
    TPMS_ENC_SCHEME_RSAES* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_ENC_SCHEME_RSAES_Unmarshal(target, buffer, size) \
      TPMS_EMPTY_Unmarshal((TPMS_EMPTY*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_ENC_SCHEME_RSAES_Marshal(
    TPMS_ENC_SCHEME_RSAES* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_ENC_SCHEME_RSAES_Marshal(source, buffer, size) \
      TPMS_EMPTY_Marshal((TPMS_EMPTY*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_ENC_SCHEME_OAEP_Unmarshal(
    TPMS_ENC_SCHEME_OAEP* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_ENC_SCHEME_OAEP_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_ENC_SCHEME_OAEP_Marshal(
    TPMS_ENC_SCHEME_OAEP* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
```

```
#  define TPMS_ENC_SCHEME_OAEP_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for ECC Key Exchange" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KEY_SCHEME_ECDH_Unmarshal(
    TPMS_KEY_SCHEME_ECDH* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KEY_SCHEME_ECDH_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_KEY_SCHEME_ECDH_Marshal(
    TPMS_KEY_SCHEME_ECDH* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KEY_SCHEME_ECDH_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KEY_SCHEME_SM2_Unmarshal(
    TPMS_KEY_SCHEME_SM2* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KEY_SCHEME_SM2_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_KEY_SCHEME_SM2_Marshal(TPMS_KEY_SCHEME_SM2* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KEY_SCHEME_SM2_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KEY_SCHEME_ECMQV_Unmarshal(
    TPMS_KEY_SCHEME_ECMQV* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KEY_SCHEME_ECMQV_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_KEY_SCHEME_ECMQV_Marshal(
    TPMS_KEY_SCHEME_ECMQV* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KEY_SCHEME_ECMQV_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for KDF Schemes" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KDF_SCHEME_MGF1_Unmarshal(
    TPMS_KDF_SCHEME_MGF1* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_MGF1_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_KDF_SCHEME_MGF1_Marshal(
    TPMS_KDF_SCHEME_MGF1* source, BYTE** buffer, INT32* size);
```

```
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_MGF1_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KDF_SCHEME_KDF1_SP800_56A_Unmarshal(
    TPMS_KDF_SCHEME_KDF1_SP800_56A* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_KDF1_SP800_56A_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_KDF_SCHEME_KDF1_SP800_56A_Marshal(
    TPMS_KDF_SCHEME_KDF1_SP800_56A* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_KDF1_SP800_56A_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KDF_SCHEME_KDF2_Unmarshal(
    TPMS_KDF_SCHEME_KDF2* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_KDF2_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_KDF_SCHEME_KDF2_Marshal(
    TPMS_KDF_SCHEME_KDF2* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_KDF2_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KDF_SCHEME_KDF1_SP800_108_Unmarshal(
    TPMS_KDF_SCHEME_KDF1_SP800_108* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_KDF1_SP800_108_Unmarshal(target, buffer, size) \
      TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_KDF_SCHEME_KDF1_SP800_108_Marshal(
    TPMS_KDF_SCHEME_KDF1_SP800_108* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_KDF_SCHEME_KDF1_SP800_108_Marshal(source, buffer, size) \
      TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_KDF_SCHEME Union" (Part 2: Structures)
TPM_RC
TPMU_KDF_SCHEME_Unmarshal(
    TPMU_KDF_SCHEME* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_KDF_SCHEME_Marshal(
    TPMU_KDF_SCHEME* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_KDF_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_KDF_SCHEME_Unmarshal(
    TPMT_KDF_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
```

```
TPMT_KDF_SCHEME_Marshal(TPMT_KDF_SCHEME* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_ALG_ASYM_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_ASYM_SCHEME_Unmarshal(
    TPMI_ALG_ASYM_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_ASYM_SCHEME_Marshal(
    TPMI_ALG_ASYM_SCHEME* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_ASYM_SCHEME_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_ASYM_SCHEME Union" (Part 2: Structures)
TPM_RC
TPMU_ASYM_SCHEME_Unmarshal(
    TPMU_ASYM_SCHEME* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_ASYM_SCHEME_Marshal(
    TPMU_ASYM_SCHEME* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMI_ALG_RSA_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_RSA_SCHEME_Unmarshal(
    TPMI_ALG_RSA_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_RSA_SCHEME_Marshal(TPMI_ALG_RSA_SCHEME* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_RSA_SCHEME_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMT_RSA_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_RSA_SCHEME_Unmarshal(
    TPMT_RSA_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_RSA_SCHEME_Marshal(TPMT_RSA_SCHEME* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_ALG_RSA_DECRYPT Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_RSA_DECRYPT_Unmarshal(
    TPMI_ALG_RSA_DECRYPT* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_RSA_DECRYPT_Marshal(
    TPMI_ALG_RSA_DECRYPT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_RSA_DECRYPT_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMT_RSA_DECRYPT Structure" (Part 2: Structures)
TPM_RC
TPMT_RSA_DECRYPT_Unmarshal(
    TPMT_RSA_DECRYPT* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_RSA_DECRYPT_Marshal(TPMT_RSA_DECRYPT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_PUBLIC_KEY_RSA Structure" (Part 2: Structures)
TPM_RC
TPM2B_PUBLIC_KEY_RSA_Unmarshal(
    TPM2B_PUBLIC_KEY_RSA* target, BYTE** buffer, INT32* size);
```

```
UINT16
TPM2B_PUBLIC_KEY_RSA_Marshal(
    TPM2B_PUBLIC_KEY_RSA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_RSA_KEY_BITS Type" (Part 2: Structures)
TPM_RC
TPMI_RSA_KEY_BITS_Unmarshal(TPMI_RSA_KEY_BITS* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_RSA_KEY_BITS_Marshal(TPMI_RSA_KEY_BITS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_RSA_KEY_BITS_Marshal(source, buffer, size) \
      TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM2B_PRIVATE_KEY_RSA Structure" (Part 2: Structures)
TPM_RC
TPM2B_PRIVATE_KEY_RSA_Unmarshal(
    TPM2B_PRIVATE_KEY_RSA* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_PRIVATE_KEY_RSA_Marshal(
    TPM2B_PRIVATE_KEY_RSA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_ECC_PARAMETER Structure" (Part 2: Structures)
TPM_RC
TPM2B_ECC_PARAMETER_Unmarshal(
    TPM2B_ECC_PARAMETER* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_ECC_PARAMETER_Marshal(TPM2B_ECC_PARAMETER* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_ECC_POINT Structure" (Part 2: Structures)
TPM_RC
TPMS_ECC_POINT_Unmarshal(TPMS_ECC_POINT* target, BYTE** buffer, INT32* size);
UINT16
TPMS_ECC_POINT_Marshal(TPMS_ECC_POINT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_ECC_POINT Structure" (Part 2: Structures)
TPM_RC
TPM2B_ECC_POINT_Unmarshal(TPM2B_ECC_POINT* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_ECC_POINT_Marshal(TPM2B_ECC_POINT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_ALG_ECC_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_ECC_SCHEME_Unmarshal(
    TPMI_ALG_ECC_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_ECC_SCHEME_Marshal(TPMI_ALG_ECC_SCHEME* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_ECC_SCHEME_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ECC_CURVE Type" (Part 2: Structures)
TPM_RC
TPMI_ECC_CURVE_Unmarshal(
    TPMI_ECC_CURVE* target, BYTE** buffer, INT32* size, BOOL flag);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ECC_CURVE_Marshal(TPMI_ECC_CURVE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ECC_CURVE_Marshal(source, buffer, size) \
      TPM_ECC_CURVE_Marshal((TPM_ECC_CURVE*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of TPMT_ECC_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_ECC_SCHEME_Unmarshal(
    TPMT_ECC_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_ECC_SCHEME_Marshal(TPMT_ECC_SCHEME* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_ALGORITHM_DETAIL_ECC Structure" (Part 2: Structures)
UINT16
TPMS_ALGORITHM_DETAIL_ECC_Marshal(
    TPMS_ALGORITHM_DETAIL_ECC* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_SIGNATURE_RSA Structure" (Part 2: Structures)
TPM_RC
TPMS_SIGNATURE_RSA_Unmarshal(TPMS_SIGNATURE_RSA* target, BYTE** buffer, INT32* size);
UINT16
TPMS_SIGNATURE_RSA_Marshal(TPMS_SIGNATURE_RSA* source, BYTE** buffer, INT32* size);

// Table "Definition of Types for Signature" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_RSASSA_Unmarshal(
    TPMS_SIGNATURE_RSASSA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_RSASSA_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_RSA_Unmarshal((TPMS_SIGNATURE_RSA*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_RSASSA_Marshal(
    TPMS_SIGNATURE_RSASSA* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_RSASSA_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_RSA_Marshal((TPMS_SIGNATURE_RSA*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_RSAPSS_Unmarshal(
    TPMS_SIGNATURE_RSAPSS* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_RSAPSS_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_RSA_Unmarshal((TPMS_SIGNATURE_RSA*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_RSAPSS_Marshal(
    TPMS_SIGNATURE_RSAPSS* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_RSAPSS_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_RSA_Marshal((TPMS_SIGNATURE_RSA*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_SIGNATURE_ECC Structure" (Part 2: Structures)
TPM_RC
TPMS_SIGNATURE_ECC_Unmarshal(TPMS_SIGNATURE_ECC* target, BYTE** buffer, INT32* size);
UINT16
TPMS_SIGNATURE_ECC_Marshal(TPMS_SIGNATURE_ECC* source, BYTE** buffer, INT32* size);

// Table "Definition of Types for TPMS_SIGNATURE_ECC" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_ECDSA_Unmarshal(
    TPMS_SIGNATURE_ECDSA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_ECDSA_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)(target), (buffer), (size))
```

```
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_ECDSA_Marshal(
    TPMS_SIGNATURE_ECDSA* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_ECDSA_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_ECDAA_Unmarshal(
    TPMS_SIGNATURE_ECDAA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_ECDAA_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_ECDAA_Marshal(
    TPMS_SIGNATURE_ECDAA* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_ECDAA_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_SM2_Unmarshal(TPMS_SIGNATURE_SM2* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_SM2_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_SM2_Marshal(TPMS_SIGNATURE_SM2* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_SM2_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_ECSCHNORR_Unmarshal(
    TPMS_SIGNATURE_ECSCHNORR* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_ECSCHNORR_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_ECSCHNORR_Marshal(
    TPMS_SIGNATURE_ECSCHNORR* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_ECSCHNORR_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_EDDSA_Unmarshal(
    TPMS_SIGNATURE_EDDSA* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_EDDSA_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_EDDSA_Marshal(
```

```
    TPMS_SIGNATURE_EDDSA* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_EDDSA_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_EDDSA_PH_Unmarshal(
    TPMS_SIGNATURE_EDDSA_PH* target, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_EDDSA_PH_Unmarshal(target, buffer, size) \
      TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)(target), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES
#if !USE_MARSHALING_DEFINES
UINT16
TPMS_SIGNATURE_EDDSA_PH_Marshal(
    TPMS_SIGNATURE_EDDSA_PH* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMS_SIGNATURE_EDDSA_PH_Marshal(source, buffer, size) \
      TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_SIGNATURE Union" (Part 2: Structures)
TPM_RC
TPMU_SIGNATURE_Unmarshal(
    TPMU_SIGNATURE* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_SIGNATURE_Marshal(
    TPMU_SIGNATURE* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_SIGNATURE Structure" (Part 2: Structures)
TPM_RC
TPMT_SIGNATURE_Unmarshal(
    TPMT_SIGNATURE* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_SIGNATURE_Marshal(TPMT_SIGNATURE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMU_ENCRYPTED_SECRET Union" (Part 2: Structures)
TPM_RC
TPMU_ENCRYPTED_SECRET_Unmarshal(
    TPMU_ENCRYPTED_SECRET* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_ENCRYPTED_SECRET_Marshal(
    TPMU_ENCRYPTED_SECRET* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPM2B_ENCRYPTED_SECRET Structure" (Part 2: Structures)
TPM_RC
TPM2B_ENCRYPTED_SECRET_Unmarshal(
    TPM2B_ENCRYPTED_SECRET* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_ENCRYPTED_SECRET_Marshal(
    TPM2B_ENCRYPTED_SECRET* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMI_ALG_PUBLIC Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_PUBLIC_Unmarshal(TPMI_ALG_PUBLIC* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_PUBLIC_Marshal(TPMI_ALG_PUBLIC* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMI_ALG_PUBLIC_Marshal(source, buffer, size) \
      TPM_ALG_ID_Marshal((TPM_ALG_ID*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_PUBLIC_ID Union" (Part 2: Structures)
TPM_RC
```

```
TPMU_PUBLIC_ID_Unmarshal(
    TPMU_PUBLIC_ID* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_PUBLIC_ID_Marshal(
    TPMU_PUBLIC_ID* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMS_KEYEDHASH_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_KEYEDHASH_PARMS_Unmarshal(
    TPMS_KEYEDHASH_PARMS* target, BYTE** buffer, INT32* size);
UINT16
TPMS_KEYEDHASH_PARMS_Marshal(
    TPMS_KEYEDHASH_PARMS* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_RSA_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_RSA_PARMS_Unmarshal(TPMS_RSA_PARMS* target, BYTE** buffer, INT32* size);
UINT16
TPMS_RSA_PARMS_Marshal(TPMS_RSA_PARMS* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_ECC_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_ECC_PARMS_Unmarshal(TPMS_ECC_PARMS* target, BYTE** buffer, INT32* size);
UINT16
TPMS_ECC_PARMS_Marshal(TPMS_ECC_PARMS* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMU_PUBLIC_PARMS Union" (Part 2: Structures)
TPM_RC
TPMU_PUBLIC_PARMS_Unmarshal(
    TPMU_PUBLIC_PARMS* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_PUBLIC_PARMS_Marshal(
    TPMU_PUBLIC_PARMS* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_PUBLIC_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMT_PUBLIC_PARMS_Unmarshal(TPMT_PUBLIC_PARMS* target, BYTE** buffer, INT32* size);
UINT16
TPMT_PUBLIC_PARMS_Marshal(TPMT_PUBLIC_PARMS* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMT_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPMT_PUBLIC_Unmarshal(TPMT_PUBLIC* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPMT_PUBLIC_Marshal(TPMT_PUBLIC* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPM2B_PUBLIC_Unmarshal(TPM2B_PUBLIC* target, BYTE** buffer, INT32* size, BOOL flag);
UINT16
TPM2B_PUBLIC_Marshal(TPM2B_PUBLIC* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_TEMPLATE Structure" (Part 2: Structures)
TPM_RC
TPM2B_TEMPLATE_Unmarshal(TPM2B_TEMPLATE* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_TEMPLATE_Marshal(TPM2B_TEMPLATE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_PRIVATE_VENDOR_SPECIFIC Structure" (Part 2: Structures)
TPM_RC
TPM2B_PRIVATE_VENDOR_SPECIFIC_Unmarshal(
    TPM2B_PRIVATE_VENDOR_SPECIFIC* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_PRIVATE_VENDOR_SPECIFIC_Marshal(
    TPM2B_PRIVATE_VENDOR_SPECIFIC* source, BYTE** buffer, INT32* size);
```

```
// Table "Definition of TPMU_SENSITIVE_COMPOSITE Union" (Part 2: Structures)
TPM_RC
TPMU_SENSITIVE_COMPOSITE_Unmarshal(
    TPMU_SENSITIVE_COMPOSITE* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_SENSITIVE_COMPOSITE_Marshal(
    TPMU_SENSITIVE_COMPOSITE* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_SENSITIVE Structure" (Part 2: Structures)
TPM_RC
TPMT_SENSITIVE_Unmarshal(TPMT_SENSITIVE* target, BYTE** buffer, INT32* size);
UINT16
TPMT_SENSITIVE_Marshal(TPMT_SENSITIVE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_SENSITIVE Structure" (Part 2: Structures)
TPM_RC
TPM2B_SENSITIVE_Unmarshal(TPM2B_SENSITIVE* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_SENSITIVE_Marshal(TPM2B_SENSITIVE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_PRIVATE Structure" (Part 2: Structures)
TPM_RC
TPM2B_PRIVATE_Unmarshal(TPM2B_PRIVATE* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_PRIVATE_Marshal(TPM2B_PRIVATE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_ID_OBJECT Structure" (Part 2: Structures)
TPM_RC
TPM2B_ID_OBJECT_Unmarshal(TPM2B_ID_OBJECT* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_ID_OBJECT_Marshal(TPM2B_ID_OBJECT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_NV_PIN_COUNTER_PARAMETERS Structure" (Part 2: Structures)
TPM_RC
TPMS_NV_PIN_COUNTER_PARAMETERS_Unmarshal(
    TPMS_NV_PIN_COUNTER_PARAMETERS* target, BYTE** buffer, INT32* size);
UINT16
TPMS_NV_PIN_COUNTER_PARAMETERS_Marshal(
    TPMS_NV_PIN_COUNTER_PARAMETERS* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMA_NV Bits" (Part 2: Structures)
TPM_RC
TPMA_NV_Unmarshal(TPMA_NV* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_NV_Marshal(TPMA_NV* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_NV_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_NV_EXP Bits" (Part 2: Structures)
TPM_RC
TPMA_NV_EXP_Unmarshal(TPMA_NV_EXP* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPMA_NV_EXP_Marshal(TPMA_NV_EXP* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPMA_NV_EXP_Marshal(source, buffer, size) \
      UINT64_Marshal((UINT64*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_NV_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPMS_NV_PUBLIC_Unmarshal(TPMS_NV_PUBLIC* target, BYTE** buffer, INT32* size);
UINT16
```

```
TPMS_NV_PUBLIC_Marshal(TPMS_NV_PUBLIC* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_NV_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPM2B_NV_PUBLIC_Unmarshal(TPM2B_NV_PUBLIC* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_NV_PUBLIC_Marshal(TPM2B_NV_PUBLIC* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_NV_PUBLIC_EXP_ATTR Structure" (Part 2: Structures)
TPM_RC
TPMS_NV_PUBLIC_EXP_ATTR_Unmarshal(
    TPMS_NV_PUBLIC_EXP_ATTR* target, BYTE** buffer, INT32* size);
UINT16
TPMS_NV_PUBLIC_EXP_ATTR_Marshal(
    TPMS_NV_PUBLIC_EXP_ATTR* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMU_NV_PUBLIC_2 Union" (Part 2: Structures)
TPM_RC
TPMU_NV_PUBLIC_2_Unmarshal(
    TPMU_NV_PUBLIC_2* target, BYTE** buffer, INT32* size, UINT32 selector);
UINT16
TPMU_NV_PUBLIC_2_Marshal(
    TPMU_NV_PUBLIC_2* source, BYTE** buffer, INT32* size, UINT32 selector);

// Table "Definition of TPMT_NV_PUBLIC_2 Structure" (Part 2: Structures)
TPM_RC
TPMT_NV_PUBLIC_2_Unmarshal(TPMT_NV_PUBLIC_2* target, BYTE** buffer, INT32* size);
UINT16
TPMT_NV_PUBLIC_2_Marshal(TPMT_NV_PUBLIC_2* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_NV_PUBLIC_2 Structure" (Part 2: Structures)
TPM_RC
TPM2B_NV_PUBLIC_2_Unmarshal(TPM2B_NV_PUBLIC_2* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_NV_PUBLIC_2_Marshal(TPM2B_NV_PUBLIC_2* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_CONTEXT_SENSITIVE Structure" (Part 2: Structures)
TPM_RC
TPM2B_CONTEXT_SENSITIVE_Unmarshal(
    TPM2B_CONTEXT_SENSITIVE* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_CONTEXT_SENSITIVE_Marshal(
    TPM2B_CONTEXT_SENSITIVE* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_CONTEXT_DATA Structure" (Part 2: Structures)
TPM_RC
TPMS_CONTEXT_DATA_Unmarshal(TPMS_CONTEXT_DATA* target, BYTE** buffer, INT32* size);
UINT16
TPMS_CONTEXT_DATA_Marshal(TPMS_CONTEXT_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM2B_CONTEXT_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_CONTEXT_DATA_Unmarshal(TPM2B_CONTEXT_DATA* target, BYTE** buffer, INT32* size);
UINT16
TPM2B_CONTEXT_DATA_Marshal(TPM2B_CONTEXT_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_CONTEXT Structure" (Part 2: Structures)
TPM_RC
TPMS_CONTEXT_Unmarshal(TPMS_CONTEXT* target, BYTE** buffer, INT32* size);
UINT16
TPMS_CONTEXT_Marshal(TPMS_CONTEXT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPMS_CREATION_DATA Structure" (Part 2: Structures)
UINT16
TPMS_CREATION_DATA_Marshal(TPMS_CREATION_DATA* source, BYTE** buffer, INT32* size);
```

```
// Table "Definition of TPM2B_CREATION_DATA Structure" (Part 2: Structures)
UINT16
TPM2B_CREATION_DATA_Marshal(TPM2B_CREATION_DATA* source, BYTE** buffer, INT32* size);

// Table "Definition of TPM_AT Constants" (Part 2: Structures)
TPM_RC
TPM_AT_Unmarshal(TPM_AT* target, BYTE** buffer, INT32* size);
#if !USE_MARSHALING_DEFINES
UINT16
TPM_AT_Marshal(TPM_AT* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_AT_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_AE Constants" (Part 2: Structures)
#if !USE_MARSHALING_DEFINES
UINT16
TPM_AE_Marshal(TPM_AE* source, BYTE** buffer, INT32* size);
#else  // !USE_MARSHALING_DEFINES
#  define TPM_AE_Marshal(source, buffer, size) \
      UINT32_Marshal((UINT32*)(source), (buffer), (size))
#endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_AC_OUTPUT Structure" (Part 2: Structures)
UINT16
TPMS_AC_OUTPUT_Marshal(TPMS_AC_OUTPUT* source, BYTE** buffer, INT32* size);

// Table "Definition of TPML_AC_CAPABILITIES Structure" (Part 2: Structures)
UINT16
TPML_AC_CAPABILITIES_Marshal(
    TPML_AC_CAPABILITIES* source, BYTE** buffer, INT32* size);

// For structures that unmarshals/marshals an array, the code calls an
// un/marshaling function to process the array of the defined type.
// This section contains the functions that perform that operation
// Array Unmarshal/Marshal for BYTE
TPM_RC
BYTE_Array_Unmarshal(BYTE* target, BYTE** buffer, INT32* size, INT32 count);
UINT16
BYTE_Array_Marshal(BYTE* source, BYTE** buffer, INT32* size, INT32 count);

// Array Unmarshal and Marshal for TPM_ALG_ID
TPM_RC
TPM_ALG_ID_Array_Unmarshal(
    TPM_ALG_ID* target, BYTE** buffer, INT32* size, INT32 count);
UINT16
TPM_ALG_ID_Array_Marshal(TPM_ALG_ID* source, BYTE** buffer, INT32* size, INT32 count);

// Array Unmarshal and Marshal for TPM_CC
TPM_RC
TPM_CC_Array_Unmarshal(TPM_CC* target, BYTE** buffer, INT32* size, INT32 count);
UINT16
TPM_CC_Array_Marshal(TPM_CC* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPM_ECC_CURVE
#if ALG_ECC
UINT16
TPM_ECC_CURVE_Array_Marshal(
    TPM_ECC_CURVE* source, BYTE** buffer, INT32* size, INT32 count);
#else  // ALG_ECC
#  define TPM_ECC_CURVE_Array_Marshal UNIMPLEMENTED_Marshal
#endif  // ALG_ECC

// Array Marshal for TPM_HANDLE
UINT16
```

```
TPM_HANDLE_Array_Marshal(TPM_HANDLE* source, BYTE** buffer, INT32* size, INT32 count);

// Array Unmarshal and Marshal for TPM2B_DIGEST
TPM_RC
TPM2B_DIGEST_Array_Unmarshal(
    TPM2B_DIGEST* target, BYTE** buffer, INT32* size, INT32 count);
UINT16
TPM2B_DIGEST_Array_Marshal(
    TPM2B_DIGEST* source, BYTE** buffer, INT32* size, INT32 count);

// Array Unmarshal and Marshal for TPM2B_VENDOR_PROPERTY
TPM_RC
TPM2B_VENDOR_PROPERTY_Array_Unmarshal(
    TPM2B_VENDOR_PROPERTY* target, BYTE** buffer, INT32* size, INT32 count);
UINT16
TPM2B_VENDOR_PROPERTY_Array_Marshal(
    TPM2B_VENDOR_PROPERTY* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPMA_CC
UINT16
TPMA_CC_Array_Marshal(TPMA_CC* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPMS_AC_OUTPUT
UINT16
TPMS_AC_OUTPUT_Array_Marshal(
    TPMS_AC_OUTPUT* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPMS_ACT_DATA
UINT16
TPMS_ACT_DATA_Array_Marshal(
    TPMS_ACT_DATA* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPMS_ALG_PROPERTY
UINT16
TPMS_ALG_PROPERTY_Array_Marshal(
    TPMS_ALG_PROPERTY* source, BYTE** buffer, INT32* size, INT32 count);

// Array Unmarshal and Marshal for TPMS_PCR_SELECTION
TPM_RC
TPMS_PCR_SELECTION_Array_Unmarshal(
    TPMS_PCR_SELECTION* target, BYTE** buffer, INT32* size, INT32 count);
UINT16
TPMS_PCR_SELECTION_Array_Marshal(
    TPMS_PCR_SELECTION* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPMS_TAGGED_PCR_SELECT
UINT16
TPMS_TAGGED_PCR_SELECT_Array_Marshal(
    TPMS_TAGGED_PCR_SELECT* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPMS_TAGGED_POLICY
UINT16
TPMS_TAGGED_POLICY_Array_Marshal(
    TPMS_TAGGED_POLICY* source, BYTE** buffer, INT32* size, INT32 count);

// Array Marshal for TPMS_TAGGED_PROPERTY
UINT16
TPMS_TAGGED_PROPERTY_Array_Marshal(
    TPMS_TAGGED_PROPERTY* source, BYTE** buffer, INT32* size, INT32 count);

// Array Unmarshal and Marshal for TPMT_HA
TPM_RC
TPMT_HA_Array_Unmarshal(
    TPMT_HA* target, BYTE** buffer, INT32* size, BOOL flag, INT32 count);
UINT16
TPMT_HA_Array_Marshal(TPMT_HA* source, BYTE** buffer, INT32* size, INT32 count);
```

```
#endif  // _MARSHAL_FP_H_
```

6.130 /tpm/include/private/prototypes/MathOnByteBuffers_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _MATH_ON_BYTE_BUFFERS_FP_H_
#define _MATH_ON_BYTE_BUFFERS_FP_H_

//*** UnsignedCmpB
// This function compare two unsigned values. The values are byte-aligned,
// big-endian numbers (e.g, a hash).
//  Return Type: int
//      1           if (a > b)
//      0           if (a = b)
//      -1          if (a < b)
LIB_EXPORT int UnsignedCompareB(UINT32      aSize,  // IN: size of a
                                const BYTE* a,      // IN: a
                                UINT32      bSize,  // IN: size of b
                                const BYTE* b       // IN: b
);

//***SignedCompareB()
// Compare two signed integers:
//  Return Type: int
//      1         if a > b
//      0         if a = b
//      -1        if a < b
int SignedCompareB(const UINT32 aSize,  // IN: size of a
                   const BYTE*  a,      // IN: a buffer
                   const UINT32 bSize,  // IN: size of b
                   const BYTE*  b       // IN: b buffer
);

//*** ModExpB
// This function is used to do modular exponentiation in support of RSA.
// The most typical uses are: 'c' = 'm'^'e' mod 'n' (RSA encrypt) and
// 'm' = 'c'^'d' mod 'n' (RSA decrypt).  When doing decryption, the 'e' parameter
// of the function will contain the private exponent 'd' instead of the public
// exponent 'e'.
//
// If the results will not fit in the provided buffer,
// an error is returned (CRYPT_ERROR_UNDERFLOW). If the results is smaller
// than the buffer, the results is de-normalized.
//
// This version is intended for use with RSA and requires that 'm' be
// less than 'n'.
//
//  Return Type: TPM_RC
//      TPM_RC_SIZE         number to exponentiate is larger than the modulus
//      TPM_RC_NO_RESULT    result will not fit into the provided buffer
//
TPM_RC
ModExpB(UINT32 cSize,  // IN: the size of the output buffer. It will
                       //     need to be the same size as the modulus
        BYTE* c,       // OUT: the buffer to receive the results
                       //     (c->size must be set to the maximum size
                       //     for the returned value)
        const UINT32 mSize,
        const BYTE*  m,  // IN: number to exponentiate
        const UINT32 eSize,
        const BYTE*  e,  // IN: power
```

```
        const UINT32 nSize,
        const BYTE*  n  // IN: modulus
);

//*** DivideB()
// Divide an integer ('n') by an integer ('d') producing a quotient ('q') and
// a remainder ('r'). If 'q' or 'r' is not needed, then the pointer to them
// may be set to NULL.
//
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT         'q' or 'r' is too small to receive the result
//
LIB_EXPORT TPM_RC DivideB(const TPM2B* n,  // IN: numerator
                          const TPM2B* d,  // IN: denominator
                          TPM2B*       q,  // OUT: quotient
                          TPM2B*       r   // OUT: remainder
);

//*** AdjustNumberB()
// Remove/add leading zeros from a number in a TPM2B. Will try to make the number
// by adding or removing leading zeros. If the number is larger than the requested
// size, it will make the number as small as possible. Setting 'requestedSize' to
// zero is equivalent to requesting that the number be normalized.
UINT16
AdjustNumberB(TPM2B* num, UINT16 requestedSize);

//*** ShiftLeft()
// This function shifts a byte buffer (a TPM2B) one byte to the left. That is,
// the most significant bit of the most significant byte is lost.
TPM2B* ShiftLeft(TPM2B* value  // IN/OUT: value to shift and shifted value out
);

#endif  // _MATH_ON_BYTE_BUFFERS_FP_H_
```

6.131 /tpm/include/private/prototypes/Memory_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  7, 2019  Time: 06:58:58PM
 */

#ifndef _MEMORY_FP_H_
#define _MEMORY_FP_H_

//*** MemoryCopy()
// This is an alias for memmove. This is used in place of memcpy because
// some of the moves may overlap and rather than try to make sure that
// memmove is used when necessary, it is always used.
void MemoryCopy(void* dest, const void* src, int sSize);

//*** MemoryEqual()
// This function indicates if two buffers have the same values in the indicated
// number of bytes.
//  Return Type: BOOL
//      TRUE(1)         all octets are the same
//      FALSE(0)        all octets are not the same
BOOL MemoryEqual(const void*  buffer1,  // IN: compare buffer1
                 const void*  buffer2,  // IN: compare buffer2
                 unsigned int size      // IN: size of bytes being compared
);

//*** MemoryCopy2B()
// This function copies a TPM2B. This can be used when the TPM2B types are
// the same or different.
//
```

```
// This function returns the number of octets in the data buffer of the TPM2B.
LIB_EXPORT INT16 MemoryCopy2B(TPM2B*       dest,    // OUT: receiving TPM2B
                              const TPM2B* source,  // IN: source TPM2B
                              unsigned int dSize  // IN: size of the receiving buffer
);

//*** MemoryConcat2B()
// This function will concatenate the buffer contents of a TPM2B to an
// the buffer contents of another TPM2B and adjust the size accordingly
//      ('a' := ('a' | 'b')).
void MemoryConcat2B(
    TPM2B*       aInOut,   // IN/OUT: destination 2B
    TPM2B*       bIn,      // IN: second 2B
    unsigned int aMaxSize  // IN: The size of aInOut.buffer (max values for
                           //     aInOut.size)
);

//*** MemoryEqual2B()
// This function will compare two TPM2B structures. To be equal, they
// need to be the same size and the buffer contexts need to be the same
// in all octets.
//  Return Type: BOOL
//      TRUE(1)         size and buffer contents are the same
//      FALSE(0)        size or buffer contents are not the same
BOOL MemoryEqual2B(const TPM2B* aIn,  // IN: compare value
                   const TPM2B* bIn   // IN: compare value
);

//*** MemorySet()
// This function will set all the octets in the specified memory range to
// the specified octet value.
// Note: A previous version had an additional parameter (dSize) that was
// intended to make sure that the destination would not be overrun. The
// problem is that, in use, all that was happening was that the value of
// size was used for dSize so there was no benefit in the extra parameter.
void MemorySet(void* dest, int value, size_t size);

//*** MemoryPad2B()
// Function to pad a TPM2B with zeros and adjust the size.
void MemoryPad2B(TPM2B* b, UINT16 newSize);

//*** Uint16ToByteArray()
// Function to write an integer to a byte array
void Uint16ToByteArray(UINT16 i, BYTE* a);

//*** Uint32ToByteArray()
// Function to write an integer to a byte array
void Uint32ToByteArray(UINT32 i, BYTE* a);

//*** Uint64ToByteArray()
// Function to write an integer to a byte array
void Uint64ToByteArray(UINT64 i, BYTE* a);

//*** ByteArrayToUint8()
// Function to write a UINT8 to a byte array. This is included for completeness
// and to allow certain macro expansions
UINT8
ByteArrayToUint8(BYTE* a);

//*** ByteArrayToUint16()
// Function to write an integer to a byte array
UINT16
ByteArrayToUint16(BYTE* a);

//*** ByteArrayToUint32()
// Function to write an integer to a byte array
```

```
UINT32
ByteArrayToUint32(BYTE* a);

//*** ByteArrayToUint64()
// Function to write an integer to a byte array
UINT64
ByteArrayToUint64(BYTE* a);

#endif  // _MEMORY_FP_H_
```

6.132 /tpm/include/private/prototypes/NvDynamic_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  7, 2020  Time: 07:15:54PM
 */

#ifndef _NV_DYNAMIC_FP_H_
#define _NV_DYNAMIC_FP_H_

//*** NvWriteNvListEnd()
// Function to write the list terminator.
NV_REF
NvWriteNvListEnd(NV_REF end);

//*** NvUpdateIndexOrderlyData()
// This function is used to cause an update of the orderly data to the NV backing
// store.
void NvUpdateIndexOrderlyData(void);

//*** NvReadIndex()
// This function is used to read the NV Index NV_INDEX. This is used so that the
// index information can be compressed and only this function would be needed
// to decompress it. Mostly, compression would only be able to save the space
// needed by the policy.
void NvReadNvIndexInfo(NV_REF    ref,     // IN: points to NV where index is located
                       NV_INDEX* nvIndex  // OUT: place to receive index data
);

//*** NvReadObject()
// This function is used to read a persistent object. This is used so that the
// object information can be compressed and only this function would be needed
// to uncompress it.
void NvReadObject(NV_REF  ref,    // IN: points to NV where index is located
                  OBJECT* object  // OUT: place to receive the object data
);

//*** NvIndexIsDefined()
// See if an index is already defined
BOOL NvIndexIsDefined(TPM_HANDLE nvHandle  // IN: Index to look for
);

//*** NvIsPlatformPersistentHandle()
// This function indicates if a handle references a persistent object in the
// range belonging to the platform.
//  Return Type: BOOL
//      TRUE(1)         handle references a platform persistent object
//                      and may reference an owner persistent object either
//      FALSE(0)        handle does not reference platform persistent object
BOOL NvIsPlatformPersistentHandle(TPM_HANDLE handle  // IN: handle
);

//*** NvIsOwnerPersistentHandle()
// This function indicates if a handle references a persistent object in the
// range belonging to the owner.
```

```
//  Return Type: BOOL
//      TRUE(1)         handle is owner persistent handle
//      FALSE(0)        handle is not owner persistent handle and may not be
//                      a persistent handle at all
BOOL NvIsOwnerPersistentHandle(TPM_HANDLE handle  // IN: handle
);

//*** NvIndexIsAccessible()
//
// This function validates that a handle references a defined NV Index and
// that the Index is currently accessible.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE           the handle points to an undefined NV Index
//                              If shEnable is CLEAR, this would include an index
//                              created using ownerAuth. If phEnableNV is CLEAR,
//                              this would include and index created using
//                              platformAuth
//      TPM_RC_NV_READLOCKED    Index is present but locked for reading and command
//                              does not write to the index
//      TPM_RC_NV_WRITELOCKED   Index is present but locked for writing and command
//                              writes to the index
TPM_RC
NvIndexIsAccessible(TPMI_RH_NV_INDEX handle  // IN: handle
);

//*** NvGetEvictObject()
// This function is used to dereference an evict object handle and get a pointer
// to the object.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE           the handle does not point to an existing
//                              persistent object
TPM_RC
NvGetEvictObject(TPM_HANDLE handle,  // IN: handle
                 OBJECT*    object   // OUT: object data
);

//*** NvIndexCacheInit()
// Function to initialize the Index cache
void NvIndexCacheInit(void);

//*** NvGetIndexData()
// This function is used to access the data in an NV Index. The data is returned
// as a byte sequence.
//
// This function requires that the NV Index be defined, and that the
// required data is within the data range.  It also requires that TPMA_NV_WRITTEN
// of the Index is SET.
void NvGetIndexData(NV_INDEX* nvIndex,  // IN: the in RAM index descriptor
                    NV_REF    locator,  // IN: where the data is located
                    UINT32    offset,   // IN: offset of NV data
                    UINT16    size,     // IN: number of octets of NV data to read
                    void*     data      // OUT: data buffer
);

//*** NvHashIndexData()
// This function adds Index data to a hash. It does this in parts to avoid large stack
// buffers.
void NvHashIndexData(HASH_STATE* hashState,  // IN: Initialized hash state
                     NV_INDEX*   nvIndex,    // IN: Index
                     NV_REF      locator,    // IN: where the data is located
                     UINT32      offset,     // IN: starting offset
                     UINT16      size        // IN: amount to hash
);

//*** NvGetUINT64Data()
// Get data in integer format of a bit or counter NV Index.
```

```
//
// This function requires that the NV Index is defined and that the NV Index
// previously has been written.
UINT64
NvGetUINT64Data(NV_INDEX* nvIndex,  // IN: the in RAM index descriptor
                NV_REF    locator   // IN: where index exists in NV
);

//*** NvWriteIndexAttributes()
// This function is used to write just the attributes of an index.
//  Return type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting so retry
//      TPM_RC_NV_UNAVAILABLE   NV is not available
TPM_RC
NvWriteIndexAttributes(TPM_HANDLE handle,
                       NV_REF     locator,    // IN: location of the index
                       TPMA_NV    attributes  // IN: attributes to write
);

//*** NvWriteIndexAuth()
// This function is used to write the authValue of an index. It is used by
// TPM2_NV_ChangeAuth()
//  Return type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting so retry
//      TPM_RC_NV_UNAVAILABLE   NV is not available
TPM_RC
NvWriteIndexAuth(NV_REF      locator,   // IN: location of the index
                 TPM2B_AUTH* authValue  // IN: the authValue to write
);

//*** NvGetIndexInfo()
// This function loads the nvIndex Info into the NV cache and returns a pointer
// to the NV_INDEX. If the returned value is zero, the index was not found.
// The 'locator' parameter, if not NULL, will be set to the offset in NV of the
// Index (the location of the handle of the Index).
//
// This function will set the index cache. If the index is orderly, the attributes
// from RAM are substituted for the attributes in the cached index
NV_INDEX* NvGetIndexInfo(TPM_HANDLE nvHandle,  // IN: the index handle
                         NV_REF*    locator    // OUT: location of the index
);

//*** NvWriteIndexData()
// This function is used to write NV index data. It is intended to be used to
// update the data associated with the default index.
//
// This function requires that the NV Index is defined, and the data is
// within the defined data range for the index.
//
// Index data is only written due to a command that modifies the data in a single
// index. There is no case where changes are made to multiple indexes data at the
// same time. Multiple attributes may be change but not multiple index data. This
// is important because we will normally be handling the index for which we have
// the cached pointer values.
//  Return type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting so retry
//      TPM_RC_NV_UNAVAILABLE   NV is not available
TPM_RC
NvWriteIndexData(NV_INDEX* nvIndex,  // IN: the description of the index
                 UINT32    offset,   // IN: offset of NV data
                 UINT32    size,     // IN: size of NV data
                 void*     data      // IN: data buffer
);

//*** NvWriteUINT64Data()
// This function to write back a UINT64 value. The various UINT64 values (bits,
```

```
// counters, and PINs) are kept in canonical format but manipulate in native
// format. This takes a native format value converts it and saves it back as
// in canonical format.
//
// This function will return the value from NV or RAM depending on the type of the
// index (orderly or not)
//
TPM_RC
NvWriteUINT64Data(NV_INDEX* nvIndex,  // IN: the description of the index
                  UINT64    intValue  // IN: the value to write
);

//*** NvGetNameByIndexHandle()
// This function is used to compute the Name of an NV Index referenced by handle.
//
// The 'name' buffer receives the bytes of the Name and the return value
// is the number of octets in the Name.
//
// This function requires that the NV Index is defined.
TPM2B_NAME* NvGetNameByIndexHandle(
    TPMI_RH_NV_INDEX handle,  // IN: handle of the index
    TPM2B_NAME*      name     // OUT: name of the index
);

//*** NvDefineIndex()
// This function is used to assign NV memory to an NV Index.
//
//  Return Type: TPM_RC
//      TPM_RC_NV_SPACE         insufficient NV space
TPM_RC
NvDefineIndex(TPMS_NV_PUBLIC* publicArea,  // IN: A template for an area to create.
              TPM2B_AUTH*     authValue    // IN: The initial authorization value
);

//*** NvAddEvictObject()
// This function is used to assign NV memory to a persistent object.
//  Return Type: TPM_RC
//      TPM_RC_NV_HANDLE        the requested handle is already in use
//      TPM_RC_NV_SPACE         insufficient NV space
TPM_RC
NvAddEvictObject(TPMI_DH_OBJECT evictHandle,  // IN: new evict handle
                 OBJECT*        object        // IN: object to be added
);

//*** NvDeleteIndex()
// This function is used to delete an NV Index.
//  Return Type: TPM_RC
//      TPM_RC_NV_UNAVAILABLE   NV is not accessible
//      TPM_RC_NV_RATE          NV is rate limiting
TPM_RC
NvDeleteIndex(NV_INDEX* nvIndex,    // IN: an in RAM index descriptor
              NV_REF    entityAddr  // IN: location in NV
);

TPM_RC
NvDeleteEvict(TPM_HANDLE handle  // IN: handle of entity to be deleted
);

//*** NvFlushHierarchy()
// This function will delete persistent objects belonging to the indicated hierarchy.
// If the storage hierarchy is selected, the function will also delete any
// NV Index defined using ownerAuth.
//  Return Type: TPM_RC
//      TPM_RC_NV_RATE           NV is unavailable because of rate limit
//      TPM_RC_NV_UNAVAILABLE    NV is inaccessible
TPM_RC
```

```
NvFlushHierarchy(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy to be flushed.
);

//*** NvSetGlobalLock()
// This function is used to SET the TPMA_NV_WRITELOCKED attribute for all
// NV indexes that have TPMA_NV_GLOBALLOCK SET. This function is use by
// TPM2_NV_GlobalWriteLock().
//  Return Type: TPM_RC
//      TPM_RC_NV_RATE          NV is unavailable because of rate limit
//      TPM_RC_NV_UNAVAILABLE   NV is inaccessible
TPM_RC
NvSetGlobalLock(void);

//*** NvCapGetPersistent()
// This function is used to get a list of handles of the persistent objects,
// starting at 'handle'.
//
// 'Handle' must be in valid persistent object handle range, but does not
// have to reference an existing persistent object.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
NvCapGetPersistent(TPMI_DH_OBJECT handle,  // IN: start handle
                   UINT32         count,   // IN: maximum number of returned handles
                   TPML_HANDLE*   handleList  // OUT: list of handle
);

//*** NvCapGetOnePersistent()
// This function returns whether a given persistent handle exists.
//
// 'Handle' must be in valid persistent object handle range.
BOOL NvCapGetOnePersistent(TPMI_DH_OBJECT handle  // IN: handle
);

//*** NvCapGetIndex()
// This function returns a list of handles of NV indexes, starting from 'handle'.
// 'Handle' must be in the range of NV indexes, but does not have to reference
// an existing NV Index.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles to report
//      NO          all the available handles has been reported
TPMI_YES_NO
NvCapGetIndex(TPMI_DH_OBJECT handle,     // IN: start handle
              UINT32         count,      // IN: max number of returned handles
              TPML_HANDLE*   handleList  // OUT: list of handle
);

//*** NvCapGetOneIndex()
// This function whether an NV index exists.
BOOL NvCapGetOneIndex(TPMI_DH_OBJECT handle);  // IN: start handle

//*** NvCapGetIndexNumber()
// This function returns the count of NV Indexes currently defined.
UINT32
NvCapGetIndexNumber(void);

//*** NvCapGetPersistentNumber()
// Function returns the count of persistent objects currently in NV memory.
UINT32
NvCapGetPersistentNumber(void);

//*** NvCapGetPersistentAvail()
// This function returns an estimate of the number of additional persistent
// objects that could be loaded into NV memory.
UINT32
```

```
NvCapGetPersistentAvail(void);

//*** NvCapGetCounterNumber()
// Get the number of defined NV Indexes that are counter indexes.
UINT32
NvCapGetCounterNumber(void);

//*** NvEntityStartup()
//  This function is called at TPM_Startup(). If the startup completes
//  a TPM Resume cycle, no action is taken. If the startup is a TPM Reset
//  or a TPM Restart, then this function will:
//  a) clear read/write lock;
//  b) reset NV Index data that has TPMA_NV_CLEAR_STCLEAR SET; and
//  c) set the lower bits in orderly counters to 1 for a non-orderly startup
//
//  It is a prerequisite that NV be available for writing before this
//  function is called.
BOOL NvEntityStartup(STARTUP_TYPE type  // IN: start up type
);

//*** NvCapGetCounterAvail()
// This function returns an estimate of the number of additional counter type
// NV indexes that can be defined.
UINT32
NvCapGetCounterAvail(void);

//*** NvFindHandle()
// this function returns the offset in NV memory of the entity associated
// with the input handle.  A value of zero indicates that handle does not
//  exist reference an existing persistent object or defined NV Index.
NV_REF
NvFindHandle(TPM_HANDLE handle);

//*** NvReadMaxCount()
// This function returns the max NV counter value.
//
UINT64
NvReadMaxCount(void);

//*** NvUpdateMaxCount()
// This function updates the max counter value to NV memory. This is just staging
// for the actual write that will occur when the NV index memory is modified.
//
void NvUpdateMaxCount(UINT64 count);

//*** NvSetMaxCount()
// This function is used at NV initialization time to set the initial value of
// the maximum counter.
void NvSetMaxCount(UINT64 value);

//*** NvGetMaxCount()
// Function to get the NV max counter value from the end-of-list marker
UINT64
NvGetMaxCount(void);

#endif  // _NV_DYNAMIC_FP_H_
```

6.133 /tpm/include/private/prototypes/NvReserved_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 04:23:27PM
 */

#ifndef _NV_RESERVED_FP_H_
```

```
#define _NV_RESERVED_FP_H_

//*** NvCheckState()
// Function to check the NV state by accessing the platform-specific function
// to get the NV state.  The result state is registered in s_NvIsAvailable
// that will be reported by NvIsAvailable.
//
// This function is called at the beginning of ExecuteCommand before any potential
// check of g_NvStatus.
void NvCheckState(void);

//*** NvCommit
// This is a wrapper for the platform function to commit pending NV writes.
BOOL NvCommit(void);

//*** NvPowerOn()
//  This function is called at _TPM_Init to initialize the NV environment.
//  Return Type: BOOL
//      TRUE(1)         all NV was initialized
//      FALSE(0)        the NV containing saved state had an error and
//                      TPM2_Startup(CLEAR) is required
BOOL NvPowerOn(void);

//*** NvManufacture()
// This function initializes the NV system at pre-install time.
//
// This function should only be called in a manufacturing environment or in a
// simulation.
//
// The layout of NV memory space is an implementation choice.
void NvManufacture(void);

//*** NvRead()
// This function is used to move reserved data from NV memory to RAM.
void NvRead(void*  outBuffer,  // OUT: buffer to receive data
            UINT32 nvOffset,   // IN: offset in NV of value
            UINT32 size        // IN: size of the value to read
);

//*** NvWrite()
// This function is used to post reserved data for writing to NV memory. Before
// the TPM completes the operation, the value will be written.
BOOL NvWrite(UINT32 nvOffset,  // IN: location in NV to receive data
             UINT32 size,      // IN: size of the data to move
             void*  inBuffer   // IN: location containing data to write
);

//*** NvUpdatePersistent()
// This function is used to update a value in the PERSISTENT_DATA structure and
// commits the value to NV.
void NvUpdatePersistent(
    UINT32 offset,  // IN: location in PERMANENT_DATA to be updated
    UINT32 size,    // IN: size of the value
    void*  buffer   // IN: the new data
);

//*** NvClearPersistent()
// This function is used to clear a persistent data entry and commit it to NV
void NvClearPersistent(UINT32 offset,  // IN: the offset in the PERMANENT_DATA
                                       //     structure to be cleared (zeroed)
                       UINT32 size     // IN: number of bytes to clear
);

//*** NvReadPersistent()
// This function reads persistent data to the RAM copy of the 'gp' structure.
void NvReadPersistent(void);
```

```
#endif  // _NV_RESERVED_FP_H_
```

6.134 /tpm/include/private/prototypes/NV_Certify_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_Certify  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_CERTIFY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_CERTIFY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   signHandle;
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPM2B_DATA       qualifyingData;
    TPMT_SIG_SCHEME  inScheme;
    UINT16           size;
    UINT16           offset;
} NV_Certify_In;

// Output structure definition
typedef struct
{
    TPM2B_ATTEST   certifyInfo;
    TPMT_SIGNATURE signature;
} NV_Certify_Out;

// Response code modifiers
#    define RC_NV_Certify_signHandle      (TPM_RC_H + TPM_RC_1)
#    define RC_NV_Certify_authHandle      (TPM_RC_H + TPM_RC_2)
#    define RC_NV_Certify_nvIndex         (TPM_RC_H + TPM_RC_3)
#    define RC_NV_Certify_qualifyingData  (TPM_RC_P + TPM_RC_1)
#    define RC_NV_Certify_inScheme        (TPM_RC_P + TPM_RC_2)
#    define RC_NV_Certify_size            (TPM_RC_P + TPM_RC_3)
#    define RC_NV_Certify_offset          (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_NV_Certify(NV_Certify_In* in, NV_Certify_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_CERTIFY_FP_H_
#endif    // CC_NV_Certify
```

6.135 /tpm/include/private/prototypes/NV_ChangeAuth_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_ChangeAuth  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_CHANGEAUTH_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_CHANGEAUTH_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_INDEX nvIndex;
    TPM2B_AUTH       newAuth;
} NV_ChangeAuth_In;

// Response code modifiers
```

```
#    define RC_NV_ChangeAuth_nvIndex (TPM_RC_H + TPM_RC_1)
#    define RC_NV_ChangeAuth_newAuth (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_NV_ChangeAuth(NV_ChangeAuth_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_CHANGEAUTH_FP_H_
#endif    // CC_NV_ChangeAuth
```

6.136 /tpm/include/private/prototypes/NV_DefineSpace2_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_DefineSpace2  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_DEFINESPACE2_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_DEFINESPACE2_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION authHandle;
    TPM2B_AUTH        auth;
    TPM2B_NV_PUBLIC_2 publicInfo;
} NV_DefineSpace2_In;

// Response code modifiers
#    define RC_NV_DefineSpace2_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_DefineSpace2_auth       (TPM_RC_P + TPM_RC_1)
#    define RC_NV_DefineSpace2_publicInfo (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_DefineSpace2(NV_DefineSpace2_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_DEFINESPACE2_FP_H_
#endif    // CC_NV_DefineSpace2
```

6.137 /tpm/include/private/prototypes/NV_DefineSpace_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_DefineSpace  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_DEFINESPACE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_DEFINESPACE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION authHandle;
    TPM2B_AUTH        auth;
    TPM2B_NV_PUBLIC   publicInfo;
} NV_DefineSpace_In;

// Response code modifiers
#    define RC_NV_DefineSpace_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_DefineSpace_auth       (TPM_RC_P + TPM_RC_1)
#    define RC_NV_DefineSpace_publicInfo (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_DefineSpace(NV_DefineSpace_In* in);
```

```
#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_DEFINESPACE_FP_H_
#endif    // CC_NV_DefineSpace
```

6.138 /tpm/include/private/prototypes/NV_Extend_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_Extend  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_EXTEND_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_EXTEND_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH     authHandle;
    TPMI_RH_NV_INDEX    nvIndex;
    TPM2B_MAX_NV_BUFFER data;
} NV_Extend_In;

// Response code modifiers
#    define RC_NV_Extend_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_Extend_nvIndex    (TPM_RC_H + TPM_RC_2)
#    define RC_NV_Extend_data       (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_NV_Extend(NV_Extend_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_EXTEND_FP_H_
#endif    // CC_NV_Extend
```

6.139 /tpm/include/private/prototypes/NV_GlobalWriteLock_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_GlobalWriteLock  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_GLOBALWRITELOCK_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_GLOBALWRITELOCK_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION authHandle;
} NV_GlobalWriteLock_In;

// Response code modifiers
#    define RC_NV_GlobalWriteLock_authHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_NV_GlobalWriteLock(NV_GlobalWriteLock_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_GLOBALWRITELOCK_FP_H_
#endif    // CC_NV_GlobalWriteLock
```

6.140 /tpm/include/private/prototypes/NV_Increment_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_Increment  // Command must be enabled
```

```
#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_INCREMENT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_INCREMENT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
} NV_Increment_In;

// Response code modifiers
#    define RC_NV_Increment_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_Increment_nvIndex    (TPM_RC_H + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_Increment(NV_Increment_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_INCREMENT_FP_H_
#endif    // CC_NV_Increment
```

6.141 /tpm/include/private/prototypes/NV_ReadLock_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_ReadLock  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READLOCK_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READLOCK_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
} NV_ReadLock_In;

// Response code modifiers
#    define RC_NV_ReadLock_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_ReadLock_nvIndex    (TPM_RC_H + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_ReadLock(NV_ReadLock_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READLOCK_FP_H_
#endif    // CC_NV_ReadLock
```

6.142 /tpm/include/private/prototypes/NV_ReadPublic2_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_ReadPublic2  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READPUBLIC2_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READPUBLIC2_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_INDEX nvIndex;
} NV_ReadPublic2_In;
```

```
// Output structure definition
typedef struct
{
    TPM2B_NV_PUBLIC_2 nvPublic;
    TPM2B_NAME        nvName;
} NV_ReadPublic2_Out;

// Response code modifiers
#    define RC_NV_ReadPublic2_nvIndex (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_NV_ReadPublic2(NV_ReadPublic2_In* in, NV_ReadPublic2_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READPUBLIC2_FP_H_
#endif    // CC_NV_ReadPublic2
```

6.143 /tpm/include/private/prototypes/NV_ReadPublic_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_ReadPublic  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READPUBLIC_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READPUBLIC_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_INDEX nvIndex;
} NV_ReadPublic_In;

// Output structure definition
typedef struct
{
    TPM2B_NV_PUBLIC nvPublic;
    TPM2B_NAME      nvName;
} NV_ReadPublic_Out;

// Response code modifiers
#    define RC_NV_ReadPublic_nvIndex (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_NV_ReadPublic(NV_ReadPublic_In* in, NV_ReadPublic_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READPUBLIC_FP_H_
#endif    // CC_NV_ReadPublic
```

6.144 /tpm/include/private/prototypes/NV_Read_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_Read  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READ_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READ_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    UINT16           size;
```

```
    UINT16          offset;
} NV_Read_In;

// Output structure definition
typedef struct
{
    TPM2B_MAX_NV_BUFFER data;
} NV_Read_Out;

// Response code modifiers
#    define RC_NV_Read_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_Read_nvIndex    (TPM_RC_H + TPM_RC_2)
#    define RC_NV_Read_size       (TPM_RC_P + TPM_RC_1)
#    define RC_NV_Read_offset     (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_Read(NV_Read_In* in, NV_Read_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_READ_FP_H_
#endif    // CC_NV_Read
```

6.145 /tpm/include/private/prototypes/NV_SetBits_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_SetBits  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_SETBITS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_SETBITS_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    UINT64           bits;
} NV_SetBits_In;

// Response code modifiers
#    define RC_NV_SetBits_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_SetBits_nvIndex    (TPM_RC_H + TPM_RC_2)
#    define RC_NV_SetBits_bits       (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_NV_SetBits(NV_SetBits_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_SETBITS_FP_H_
#endif    // CC_NV_SetBits
```

6.146 /tpm/include/private/prototypes/NV_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _NV_SPT_FP_H_
#define _NV_SPT_FP_H_

//*** NvReadAccessChecks()
// Common routine for validating a read
// Used by TPM2_NV_Read, TPM2_NV_ReadLock and TPM2_PolicyNV
```

```
//  Return Type: TPM_RC
//      TPM_RC_NV_AUTHORIZATION     authHandle is not allowed to authorize read
//                                  of the index
//      TPM_RC_NV_LOCKED            Read locked
//      TPM_RC_NV_UNINITIALIZED     Try to read an uninitialized index
//
TPM_RC
NvReadAccessChecks(TPM_HANDLE authHandle,  // IN: the handle that provided the
                                           //     authorization
                   TPM_HANDLE nvHandle,   // IN: the handle of the NV index to be read
                   TPMA_NV    attributes  // IN: the attributes of 'nvHandle'
);

//*** NvWriteAccessChecks()
// Common routine for validating a write
// Used by TPM2_NV_Write, TPM2_NV_Increment, TPM2_SetBits, and TPM2_NV_WriteLock
//  Return Type: TPM_RC
//      TPM_RC_NV_AUTHORIZATION     Authorization fails
//      TPM_RC_NV_LOCKED            Write locked
//
TPM_RC
NvWriteAccessChecks(
    TPM_HANDLE authHandle,  // IN: the handle that provided the
                            //     authorization
    TPM_HANDLE nvHandle,    // IN: the handle of the NV index to be written
    TPMA_NV    attributes   // IN: the attributes of 'nvHandle'
);

//*** NvClearOrderly()
// This function is used to cause gp.orderlyState to be cleared to the
// non-orderly state.
TPM_RC
NvClearOrderly(void);

//*** NvIsPinPassIndex()
// Function to check to see if an NV index is a PIN Pass Index
//  Return Type: BOOL
//      TRUE(1)         is pin pass
//      FALSE(0)        is not pin pass
BOOL NvIsPinPassIndex(TPM_HANDLE index  // IN: Handle to check
);

//*** NvGetIndexName()
// This function computes the Name of an index
// The 'name' buffer receives the bytes of the Name and the return value
// is the number of octets in the Name.
//
// This function requires that the NV Index is defined.
TPM2B_NAME* NvGetIndexName(
    NV_INDEX* nvIndex,  // IN: the index over which the name is to be
                        //     computed
    TPM2B_NAME* name    // OUT: name of the index
);

//*** NvPublic2FromNvPublic()
// This function converts a legacy-form NV public (TPMS_NV_PUBLIC) into the
// generalized TPMT_NV_PUBLIC_2 tagged-union representation.
TPM_RC NvPublic2FromNvPublic(
    TPMS_NV_PUBLIC*   nvPublic,  // IN: the source S-form NV public area
    TPMT_NV_PUBLIC_2* nvPublic2  // OUT: the T-form NV public area to populate
);

//*** NvPublicFromNvPublic2()
// This function converts a tagged-union NV public (TPMT_NV_PUBLIC_2) into the
// legacy TPMS_NV_PUBLIC representation. This is a lossy conversion: any
// bits in the extended area of the attributes are lost, and the Name cannot be
```

```
// computed based on it.
TPM_RC NvPublicFromNvPublic2(
    TPMT_NV_PUBLIC_2* nvPublic2,  // IN: the source T-form NV public area
    TPMS_NV_PUBLIC*   nvPublic    // OUT: the S-form NV public area to populate
);

//*** NvDefineSpace()
// This function combines the common functionality of TPM2_NV_DefineSpace and
// TPM2_NV_DefineSpace2.
TPM_RC NvDefineSpace(TPMI_RH_PROVISION authHandle,
                     TPM2B_AUTH*       auth,
                     TPMS_NV_PUBLIC*   publicInfo,
                     TPM_RC            blameAuthHandle,
                     TPM_RC            blameAuth,
                     TPM_RC            blamePublic);

#endif  // _NV_SPT_FP_H_
```

6.147 /tpm/include/private/prototypes/NV_UndefineSpaceSpecial_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_UndefineSpaceSpecial  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_UNDEFINESPACESPECIAL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_UNDEFINESPACESPECIAL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_DEFINED_INDEX nvIndex;
    TPMI_RH_PLATFORM         platform;
} NV_UndefineSpaceSpecial_In;

// Response code modifiers
#    define RC_NV_UndefineSpaceSpecial_nvIndex  (TPM_RC_H + TPM_RC_1)
#    define RC_NV_UndefineSpaceSpecial_platform (TPM_RC_H + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_UndefineSpaceSpecial(NV_UndefineSpaceSpecial_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_UNDEFINESPACESPECIAL_FP_H_
#endif    // CC_NV_UndefineSpaceSpecial
```

6.148 /tpm/include/private/prototypes/NV_UndefineSpace_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_UndefineSpace  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_UNDEFINESPACE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_UNDEFINESPACE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION        authHandle;
    TPMI_RH_NV_DEFINED_INDEX nvIndex;
} NV_UndefineSpace_In;

// Response code modifiers
#    define RC_NV_UndefineSpace_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_UndefineSpace_nvIndex    (TPM_RC_H + TPM_RC_2)
```

```
// Function prototype
TPM_RC
TPM2_NV_UndefineSpace(NV_UndefineSpace_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_UNDEFINESPACE_FP_H_
#endif    // CC_NV_UndefineSpace
```

6.149 /tpm/include/private/prototypes/NV_WriteLock_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_WriteLock  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_WRITELOCK_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_WRITELOCK_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
} NV_WriteLock_In;

// Response code modifiers
#    define RC_NV_WriteLock_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_WriteLock_nvIndex    (TPM_RC_H + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_WriteLock(NV_WriteLock_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_WRITELOCK_FP_H_
#endif    // CC_NV_WriteLock
```

6.150 /tpm/include/private/prototypes/NV_Write_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_NV_Write  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_WRITE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_WRITE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH     authHandle;
    TPMI_RH_NV_INDEX    nvIndex;
    TPM2B_MAX_NV_BUFFER data;
    UINT16              offset;
} NV_Write_In;

// Response code modifiers
#    define RC_NV_Write_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_NV_Write_nvIndex    (TPM_RC_H + TPM_RC_2)
#    define RC_NV_Write_data       (TPM_RC_P + TPM_RC_1)
#    define RC_NV_Write_offset     (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_NV_Write(NV_Write_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_NV_WRITE_FP_H_
```

```
#endif    // CC_NV_Write
```

6.151 /tpm/include/private/prototypes/ObjectChangeAuth_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ObjectChangeAuth  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_OBJECTCHANGEAUTH_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_OBJECTCHANGEAUTH_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT objectHandle;
    TPMI_DH_OBJECT parentHandle;
    TPM2B_AUTH     newAuth;
} ObjectChangeAuth_In;

// Output structure definition
typedef struct
{
    TPM2B_PRIVATE outPrivate;
} ObjectChangeAuth_Out;

// Response code modifiers
#    define RC_ObjectChangeAuth_objectHandle (TPM_RC_H + TPM_RC_1)
#    define RC_ObjectChangeAuth_parentHandle (TPM_RC_H + TPM_RC_2)
#    define RC_ObjectChangeAuth_newAuth      (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ObjectChangeAuth(ObjectChangeAuth_In* in, ObjectChangeAuth_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_OBJECTCHANGEAUTH_FP_H_
#endif    // CC_ObjectChangeAuth
```

6.152 /tpm/include/private/prototypes/Object_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _OBJECT_FP_H_
#define _OBJECT_FP_H_

//*** ObjectFlush()
// This function marks an object slot as available.
// Since there is no checking of the input parameters, it should be used
// judiciously.
// Note: This could be converted to a macro.
void ObjectFlush(OBJECT* object);

//*** ObjectSetInUse()
// This access function sets the occupied attribute of an object slot.
void ObjectSetInUse(OBJECT* object);

//*** ObjectStartup()
// This function is called at TPM2_Startup() to initialize the object subsystem.
BOOL ObjectStartup(void);

//*** ObjectCleanupEvict()
//
```

```
// In this implementation, a persistent object is moved from NV into an object slot
// for processing. It is flushed after command execution. This function is called
// from ExecuteCommand().
void ObjectCleanupEvict(void);

//*** IsObjectPresent()
// This function checks to see if a transient handle references a loaded
// object.  This routine should not be called if the handle is not a
// transient handle. The function validates that the handle is in the
// implementation-dependent allowed in range for loaded transient objects.
//  Return Type: BOOL
//      TRUE(1)         handle references a loaded object
//      FALSE(0)        handle is not an object handle, or it does not
//                      reference to a loaded object
BOOL IsObjectPresent(TPMI_DH_OBJECT handle  // IN: handle to be checked
);

//*** ObjectIsSequence()
// This function is used to check if the object is a sequence object. This function
// should not be called if the handle does not reference a loaded object.
//  Return Type: BOOL
//      TRUE(1)         object is an HMAC, hash, or event sequence object
//      FALSE(0)        object is not an HMAC, hash, or event sequence object
BOOL ObjectIsSequence(OBJECT* object  // IN: handle to be checked
);

//*** HandleToObject()
// This function is used to find the object structure associated with a handle.
//
// This function requires that 'handle' references a loaded object or a permanent
// handle.
OBJECT* HandleToObject(TPMI_DH_OBJECT handle  // IN: handle of the object
);

//*** GetQualifiedName()
// This function returns the Qualified Name of the object. In this implementation,
// the Qualified Name is computed when the object is loaded and is saved in the
// internal representation of the object. The alternative would be to retain the
// Name of the parent and compute the QN when needed. This would take the same
// amount of space so it is not recommended that the alternate be used.
//
// This function requires that 'handle' references a loaded object.
void GetQualifiedName(TPMI_DH_OBJECT handle,     // IN: handle of the object
                      TPM2B_NAME* qualifiedName  // OUT: qualified name of the object
);

//*** GetHierarchy()
// This function returns the handle of the hierarchy to which a handle belongs.
//
// This function requires that 'handle' references a loaded object.
TPMI_RH_HIERARCHY
GetHierarchy(TPMI_DH_OBJECT handle  // IN :object handle
);

//*** FindEmptyObjectSlot()
// This function finds an open object slot, if any. It will clear the attributes
// but will not set the occupied attribute. This is so that a slot may be used
// and discarded if everything does not go as planned.
//  Return Type: OBJECT *
//      NULL        no open slot found
//      != NULL     pointer to available slot
OBJECT* FindEmptyObjectSlot(TPMI_DH_OBJECT* handle  // OUT: (optional)
);

//*** ObjectAllocateSlot()
// This function is used to allocate a slot in internal object array.
```

```
OBJECT* ObjectAllocateSlot(TPMI_DH_OBJECT* handle  // OUT: handle of allocated object
);

//*** ObjectSetLoadedAttributes()
// This function sets the internal attributes for a loaded object. It is called to
// finalize the OBJECT attributes (not the TPMA_OBJECT attributes) for a loaded
// object.
void ObjectSetLoadedAttributes(OBJECT* object,  // IN: object attributes to finalize
                               TPM_HANDLE parentHandle  // IN: the parent handle
);

//*** ObjectLoad()
// Common function to load a non-primary object (i.e., either an Ordinary Object,
// or an External Object). A loaded object has its public area validated
// (unless its 'nameAlg' is TPM_ALG_NULL). If a sensitive part is loaded, it is
// verified to be correct and if both public and sensitive parts are loaded, then
// the cryptographic binding between the objects is validated. This function does
// not cause the allocated slot to be marked as in use.
TPM_RC
ObjectLoad(OBJECT* object,           // IN: pointer to object slot
                                     //     object
           OBJECT*      parent,      // IN: (optional) the parent object
           TPMT_PUBLIC* publicArea,  // IN: public area to be installed in the object
           TPMT_SENSITIVE* sensitive,  // IN: (optional) sensitive area to be
                                       //      installed in the object
           TPM_RC blamePublic,         // IN: parameter number to associate with the
                                       //     publicArea errors
           TPM_RC blameSensitive,      // IN: parameter number to associate with the
                                       //     sensitive area errors
           TPM2B_NAME* name            // IN: (optional)
);

#if CC_HMAC_Start || CC_MAC_Start
//*** ObjectCreateHMACSequence()
// This function creates an internal HMAC sequence object.
//  Return Type: TPM_RC
//      TPM_RC_OBJECT_MEMORY        if there is no free slot for an object
TPM_RC
ObjectCreateHMACSequence(
    TPMI_ALG_HASH   hashAlg,    // IN: hash algorithm
    OBJECT*         keyObject,  // IN: the object containing the HMAC key
    TPM2B_AUTH*     auth,       // IN: authValue
    TPMI_DH_OBJECT* newHandle   // OUT: HMAC sequence object handle
);
#endif

//*** ObjectCreateHashSequence()
// This function creates a hash sequence object.
//  Return Type: TPM_RC
//      TPM_RC_OBJECT_MEMORY        if there is no free slot for an object
TPM_RC
ObjectCreateHashSequence(TPMI_ALG_HASH   hashAlg,   // IN: hash algorithm
                         TPM2B_AUTH*     auth,      // IN: authValue
                         TPMI_DH_OBJECT* newHandle  // OUT: sequence object handle
);

//*** ObjectCreateEventSequence()
// This function creates an event sequence object.
//  Return Type: TPM_RC
//      TPM_RC_OBJECT_MEMORY        if there is no free slot for an object
TPM_RC
ObjectCreateEventSequence(TPM2B_AUTH*     auth,      // IN: authValue
                          TPMI_DH_OBJECT* newHandle  // OUT: sequence object handle
);

//*** ObjectTerminateEvent()
```

```
// This function is called to close out the event sequence and clean up the hash
// context states.
void ObjectTerminateEvent(void);

//*** ObjectContextLoad()
// This function loads an object from a saved object context.
//  Return Type: OBJECT *
//      NULL        if there is no free slot for an object
//      != NULL     points to the loaded object
OBJECT* ObjectContextLoad(
    ANY_OBJECT_BUFFER* object,  // IN: pointer to object structure in saved
                                //     context
    TPMI_DH_OBJECT* handle      // OUT: object handle
);

//*** FlushObject()
// This function frees an object slot.
//
// This function requires that the object is loaded.
void FlushObject(TPMI_DH_OBJECT handle  // IN: handle to be freed
);

//*** ObjectFlushHierarchy()
// This function is called to flush all the loaded transient objects associated
// with a hierarchy when the hierarchy is disabled.
void ObjectFlushHierarchy(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy to be flush
);

//*** ObjectLoadEvict()
// This function loads a persistent object into a transient object slot.
//
// This function requires that 'handle' is associated with a persistent object.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE               the persistent object does not exist
//                                  or the associated hierarchy is disabled.
//      TPM_RC_OBJECT_MEMORY        no object slot
TPM_RC
ObjectLoadEvict(TPM_HANDLE* handle,  // IN:OUT: evict object handle.  If success, it
                                     // will be replace by the loaded object handle
                COMMAND_INDEX commandIndex  // IN: the command being processed
);

//*** ObjectComputeName()
// This does the name computation from a public area (can be marshaled or not).
TPM2B_NAME* ObjectComputeName(UINT32     size,  // IN: the size of the area to digest
                              BYTE*      publicArea,  // IN: the public area to digest
                              TPM_ALG_ID nameAlg,     // IN: the hash algorithm to use
                              TPM2B_NAME* name        // OUT: Computed name
);

//*** PublicMarshalAndComputeName()
// This function computes the Name of an object from its public area.
TPM2B_NAME* PublicMarshalAndComputeName(
    TPMT_PUBLIC* publicArea,  // IN: public area of an object
    TPM2B_NAME*  name         // OUT: name of the object
);

//*** ComputeQualifiedName()
// This function computes the qualified name of an object.
void ComputeQualifiedName(
    TPM_HANDLE  parentHandle,  // IN: parent's handle
    TPM_ALG_ID  nameAlg,       // IN: name hash
    TPM2B_NAME* name,          // IN: name of the object
    TPM2B_NAME* qualifiedName  // OUT: qualified name of the object
);
```

```
//*** ObjectIsStorage()
// This function determines if an object has the attributes associated
// with a parent. A parent is an asymmetric or symmetric block cipher key
// that has its 'restricted' and 'decrypt' attributes SET, and 'sign' CLEAR.
//  Return Type: BOOL
//      TRUE(1)         object is a storage key
//      FALSE(0)        object is not a storage key
BOOL ObjectIsStorage(TPMI_DH_OBJECT handle  // IN: object handle
);

//*** ObjectCapGetLoaded()
// This function returns a list of handles of loaded object, starting from
// 'handle'. 'Handle' must be in the range of valid transient object handles,
// but does not have to be the handle of a loaded transient object.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
ObjectCapGetLoaded(TPMI_DH_OBJECT handle,     // IN: start handle
                   UINT32         count,      // IN: count of returned handles
                   TPML_HANDLE*   handleList  // OUT: list of handle
);

//*** ObjectCapGetOneLoaded()
// This function returns whether a handle is loaded.
BOOL ObjectCapGetOneLoaded(TPMI_DH_OBJECT handle  // IN: handle
);

//*** ObjectCapGetTransientAvail()
// This function returns an estimate of the number of additional transient
// objects that could be loaded into the TPM.
UINT32
ObjectCapGetTransientAvail(void);

//*** ObjectGetPublicAttributes()
// Returns the attributes associated with an object handles.
TPMA_OBJECT
ObjectGetPublicAttributes(TPM_HANDLE handle);

OBJECT_ATTRIBUTES
ObjectGetProperties(TPM_HANDLE handle);

#endif  // _OBJECT_FP_H_
```

6.153 /tpm/include/private/prototypes/Object_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  7, 2020  Time: 07:06:44PM
 */

#ifndef _OBJECT_SPT_FP_H_
#define _OBJECT_SPT_FP_H_

//*** AdjustAuthSize()
// This function will validate that the input authValue is no larger than the
// digestSize for the nameAlg. It will then pad with zeros to the size of the
// digest.
BOOL AdjustAuthSize(TPM2B_AUTH*   auth,    // IN/OUT: value to adjust
                    TPMI_ALG_HASH nameAlg  // IN:
);

//*** AreAttributesForParent()
// This function is called by create, load, and import functions.
//
```

```
// Note: The 'isParent' attribute is SET when an object is loaded and it has
// attributes that are suitable for a parent object.
//  Return Type: BOOL
//      TRUE(1)         properties are those of a parent
//      FALSE(0)        properties are not those of a parent
BOOL ObjectIsParent(OBJECT* parentObject  // IN: parent handle
);

//*** CreateChecks()
// Attribute checks that are unique to creation.
// If parentObject is not NULL, then this function checks the object's
// attributes as an Ordinary or Derived Object with the given parent.
// If parentObject is NULL, and primaryHandle is not 0, then this function
// checks the object's attributes as a Primary Object in the given hierarchy.
// If parentObject is NULL, and primaryHandle is 0, then this function checks
// the object's attributes as an External Object.
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       sensitiveDataOrigin is not consistent with the
//                              object type
//      other                   returns from PublicAttributesValidation()
TPM_RC
CreateChecks(OBJECT*           parentObject,
             TPMI_RH_HIERARCHY primaryHierarchy,
             TPMT_PUBLIC*      publicArea,
             UINT16            sensitiveDataSize);

//*** SchemeChecks
// This function is called by TPM2_LoadExternal() and PublicAttributesValidation().
// This function validates the schemes in the public area of an object.
//  Return Type: TPM_RC
//      TPM_RC_HASH         non-duplicable storage key and its parent have different
//                          name algorithm
//      TPM_RC_KDF          incorrect KDF specified for decrypting keyed hash object
//      TPM_RC_KEY          invalid key size values in an asymmetric key public area
//      TPM_RCS_SCHEME       inconsistent attributes 'decrypt', 'sign', 'restricted'
//                          and key's scheme ID; or hash algorithm is inconsistent
//                          with the scheme ID for keyed hash object
//      TPM_RC_SYMMETRIC    a storage key with no symmetric algorithm specified; or
//                          non-storage key with symmetric algorithm different from
//                          TPM_ALG_NULL
TPM_RC
SchemeChecks(OBJECT*      parentObject,  // IN: parent (null if primary seed)
             TPMT_PUBLIC* publicArea     // IN: public area of the object
);

//*** PublicAttributesValidation()
// This function validates the values in the public area of an object.
// This function is used in the processing of TPM2_Create, TPM2_CreatePrimary,
// TPM2_CreateLoaded(), TPM2_Load(),  TPM2_Import(), and TPM2_LoadExternal().
// For TPM2_Import() this is only used if the new parent has fixedTPM SET. For
// TPM2_LoadExternal(), this is not used for a public-only key.
// If parentObject is not NULL, then primaryHandle is not used.
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   'fixedTPM', 'fixedParent', or 'encryptedDuplication'
//                          attributes are inconsistent between themselves or with
//                          those of the parent object;
//                          inconsistent 'restricted', 'decrypt' and 'sign'
//                          attributes;
//                          attempt to inject sensitive data for an asymmetric key;
//                          attempt to create a symmetric cipher key that is not
//                          a decryption key
//      TPM_RC_HASH         nameAlg is TPM_ALG_NULL
//      TPM_RC_SIZE         'authPolicy' size does not match digest size of the name
//                          algorithm in 'publicArea'
//   other                  returns from SchemeChecks()
TPM_RC
```

```
PublicAttributesValidation(
    // IN: input parent object (if ordinary or derived object; NULL otherwise)
    OBJECT* parentObject,
    // IN: hierarchy (if primary object; 0 otherwise)
    TPMI_RH_HIERARCHY primaryHierarchy,
    // IN: public area of the object
    TPMT_PUBLIC* publicArea);

//*** FillInCreationData()
// Fill in creation data for an object.
//  Return Type: void
void FillInCreationData(
    TPMI_DH_OBJECT       parentHandle,   // IN: handle of parent
    TPMI_ALG_HASH        nameHashAlg,    // IN: name hash algorithm
    TPML_PCR_SELECTION*  creationPCR,    // IN: PCR selection
    TPM2B_DATA*          outsideData,    // IN: outside data
    TPM2B_CREATION_DATA* outCreation,    // OUT: creation data for output
    TPM2B_DIGEST*        creationDigest  // OUT: creation digest
);

//*** GetSeedForKDF()
// Get a seed for KDF.  The KDF for encryption and HMAC key use the same seed.
const TPM2B* GetSeedForKDF(OBJECT* protector  // IN: the protector handle
);

//*** ProduceOuterWrap()
// This function produce outer wrap for a buffer containing the sensitive data.
// It requires the sensitive data being marshaled to the outerBuffer, with the
// leading bytes reserved for integrity hash.  If iv is used, iv space should
// be reserved at the beginning of the buffer.  It assumes the sensitive data
// starts at address (outerBuffer + integrity size @).
// This function:
//  a) adds IV before sensitive area if required;
//  b) encrypts sensitive data with IV or a NULL IV as required;
//  c) adds HMAC integrity at the beginning of the buffer; and
//  d) returns the total size of blob with outer wrap.
UINT16
ProduceOuterWrap(OBJECT* protector,   // IN: The handle of the object that provides
                                      //     protection.  For object, it is parent
                                      //     handle. For credential, it is the handle
                                      //     of encrypt object.
                 TPM2B*     name,     // IN: the name of the object
                 TPM_ALG_ID hashAlg,  // IN: hash algorithm for outer wrap
                 TPM2B*     seed,     // IN: an external seed may be provided for
                                      //     duplication blob. For non duplication
                                      //     blob, this parameter should be NULL
                 BOOL   useIV,        // IN: indicate if an IV is used
                 UINT16 dataSize,     // IN: the size of sensitive data, excluding the
                                      //     leading integrity buffer size or the
                                      //     optional iv size
                 BYTE* outerBuffer    // IN/OUT: outer buffer with sensitive data in
                                      //     it
);

//*** UnwrapOuter()
// This function remove the outer wrap of a blob containing sensitive data
// This function:
//  a) checks integrity of outer blob; and
//  b) decrypts the outer blob.
//
//  Return Type: TPM_RC
//      TPM_RCS_INSUFFICIENT     error during sensitive data unmarshaling
//      TPM_RCS_INTEGRITY        sensitive data integrity is broken
//      TPM_RCS_SIZE             error during sensitive data unmarshaling
//      TPM_RCS_VALUE            IV size for CFB does not match the encryption
//                               algorithm block size
```

```
TPM_RC
UnwrapOuter(OBJECT* protector,    // IN: The object that provides
                                  //     protection.  For object, it is parent
                                  //     handle. For credential, it is the
                                  //     encrypt object.
            TPM2B*     name,      // IN: the name of the object
            TPM_ALG_ID hashAlg,   // IN: hash algorithm for outer wrap
            TPM2B*     seed,      // IN: an external seed may be provided for
                                  //     duplication blob. For non duplication
                                  //     blob, this parameter should be NULL.
            BOOL   useIV,         // IN: indicates if an IV is used
            UINT16 dataSize,      // IN: size of sensitive data in outerBuffer,
                                  //     including the leading integrity buffer
                                  //     size, and an optional iv area
            BYTE* outerBuffer     // IN/OUT: sensitive data
);

//*** SensitiveToPrivate()
// This function prepare the private blob for off the chip storage
// This function:
//  a) marshals TPM2B_SENSITIVE structure into the buffer of TPM2B_PRIVATE
//  b) applies encryption to the sensitive area; and
//  c) applies outer integrity computation.
void SensitiveToPrivate(
    TPMT_SENSITIVE* sensitive,  // IN: sensitive structure
    TPM2B_NAME*     name,       // IN: the name of the object
    OBJECT*         parent,     // IN: The parent object
    TPM_ALG_ID      nameAlg,    // IN: hash algorithm in public area.  This
                                //     parameter is used when parentHandle is
                                //     NULL, in which case the object is
                                //     temporary.
    TPM2B_PRIVATE* outPrivate   // OUT: output private structure
);

//*** PrivateToSensitive()
// Unwrap a input private area.  Check the integrity, decrypt and retrieve data
// to a sensitive structure.
// This function:
//  a) checks the integrity HMAC of the input private area;
//  b) decrypts the private buffer; and
//  c) unmarshals TPMT_SENSITIVE structure into the buffer of TPMT_SENSITIVE.
//  Return Type: TPM_RC
//      TPM_RCS_INTEGRITY       if the private area integrity is bad
//      TPM_RC_SENSITIVE        unmarshal errors while unmarshaling TPMS_ENCRYPT
//                              from input private
//      TPM_RCS_SIZE            error during sensitive data unmarshaling
//      TPM_RCS_VALUE           outer wrapper does not have an iV of the correct
//                              size
TPM_RC
PrivateToSensitive(TPM2B*     inPrivate,  // IN: input private structure
                   TPM2B*     name,       // IN: the name of the object
                   OBJECT*    parent,     // IN: parent object
                   TPM_ALG_ID nameAlg,    // IN: hash algorithm in public area.  It is
                   //     passed separately because we only pass
                   //     name, rather than the whole public area
                   //     of the object.  This parameter is used in
                   //     the following two cases: 1. primary
                   //     objects. 2. duplication blob with inner
                   //     wrap.  In other cases, this parameter
                   //     will be ignored
                   TPMT_SENSITIVE* sensitive  // OUT: sensitive structure
);

//*** SensitiveToDuplicate()
// This function prepare the duplication blob from the sensitive area.
// This function:
```

```
//  a) marshals TPMT_SENSITIVE structure into the buffer of TPM2B_PRIVATE;
//  b) applies inner wrap to the sensitive area if required; and
//  c) applies outer wrap if required.
void SensitiveToDuplicate(
    TPMT_SENSITIVE* sensitive,    // IN: sensitive structure
    TPM2B*          name,         // IN: the name of the object
    OBJECT*         parent,       // IN: The new parent object
    TPM_ALG_ID      nameAlg,      // IN: hash algorithm in public area. It
                                  //     is passed separately because we
                                  //     only pass name, rather than the
                                  //     whole public area of the object.
    TPM2B* seed,                  // IN: the external seed. If external
                                  //     seed is provided with size of 0,
                                  //     no outer wrap should be applied
                                  //     to duplication blob.
    TPMT_SYM_DEF_OBJECT* symDef,  // IN: Symmetric key definition. If the
                                  //     symmetric key algorithm is NULL,
                                  //     no inner wrap should be applied.
    TPM2B_DATA* innerSymKey,      // IN/OUT: a symmetric key may be
                                  //     provided to encrypt the inner
                                  //     wrap of a duplication blob. May
                                  //     be generated here if needed.
    TPM2B_PRIVATE* outPrivate     // OUT: output private structure
);

//*** DuplicateToSensitive()
// Unwrap a duplication blob.  Check the integrity, decrypt and retrieve data
// to a sensitive structure.
// This function:
//  a) checks the integrity HMAC of the input private area;
//  b) decrypts the private buffer; and
//  c) unmarshals TPMT_SENSITIVE structure into the buffer of TPMT_SENSITIVE.
//
//  Return Type: TPM_RC
//      TPM_RC_INSUFFICIENT      unmarshaling sensitive data from 'inPrivate' failed
//      TPM_RC_INTEGRITY         'inPrivate' data integrity is broken
//      TPM_RC_SIZE              unmarshaling sensitive data from 'inPrivate' failed
TPM_RC
DuplicateToSensitive(
    TPM2B*     inPrivate,         // IN: input private structure
    TPM2B*     name,              // IN: the name of the object
    OBJECT*    parent,            // IN: the parent
    TPM_ALG_ID nameAlg,           // IN: hash algorithm in public area.
    TPM2B*     seed,              // IN: an external seed may be provided.
                                  //     If external seed is provided with
                                  //     size of 0, no outer wrap is
                                  //     applied
    TPMT_SYM_DEF_OBJECT* symDef,  // IN: Symmetric key definition. If the
                                  //     symmetric key algorithm is NULL,
                                  //     no inner wrap is applied
    TPM2B* innerSymKey,           // IN: a symmetric key may be provided
                                  //     to decrypt the inner wrap of a
                                  //     duplication blob.
    TPMT_SENSITIVE* sensitive     // OUT: sensitive structure
);

//*** SecretToCredential()
// This function prepare the credential blob from a secret (a TPM2B_DIGEST)
// This function:
//  a) marshals TPM2B_DIGEST structure into the buffer of TPM2B_ID_OBJECT;
//  b) encrypts the private buffer, excluding the leading integrity HMAC area;
//  c) computes integrity HMAC and append to the beginning of the buffer; and
//  d) sets the total size of TPM2B_ID_OBJECT buffer.
void SecretToCredential(TPM2B_DIGEST*    secret,      // IN: secret information
                        TPM2B*           name,        // IN: the name of the object
                        TPM2B*           seed,        // IN: an external seed.
```

```
                      OBJECT*          protector,   // IN: the protector
                      TPM2B_ID_OBJECT* outIDObject  // OUT: output credential
);

//*** CredentialToSecret()
// Unwrap a credential.  Check the integrity, decrypt and retrieve data
// to a TPM2B_DIGEST structure.
// This function:
//  a) checks the integrity HMAC of the input credential area;
//  b) decrypts the credential buffer; and
//  c) unmarshals TPM2B_DIGEST structure into the buffer of TPM2B_DIGEST.
//
//  Return Type: TPM_RC
//      TPM_RC_INSUFFICIENT      error during credential unmarshaling
//      TPM_RC_INTEGRITY         credential integrity is broken
//      TPM_RC_SIZE              error during credential unmarshaling
//      TPM_RC_VALUE             IV size does not match the encryption algorithm
//                               block size
TPM_RC
CredentialToSecret(TPM2B*        inIDObject,  // IN: input credential blob
                   TPM2B*        name,        // IN: the name of the object
                   TPM2B*        seed,        // IN: an external seed.
                   OBJECT*       protector,   // IN: the protector
                   TPM2B_DIGEST* secret       // OUT: secret information
);

//*** MemoryRemoveTrailingZeros()
// This function is used to adjust the length of an authorization value.
// It adjusts the size of the TPM2B so that it does not include octets
// at the end of the buffer that contain zero.
//
// This function returns the number of non-zero octets in the buffer.
UINT16
MemoryRemoveTrailingZeros(TPM2B_AUTH* auth  // IN/OUT: value to adjust
);

//*** SetLabelAndContext()
// This function sets the label and context for a derived key. It is possible
// that 'label' or 'context' can end up being an Empty Buffer.
TPM_RC
SetLabelAndContext(TPMS_DERIVE* labelContext,       // IN/OUT: the recovered label and
                                                    //      context
                   TPM2B_SENSITIVE_DATA* sensitive  // IN: the sensitive data
);

//*** UnmarshalToPublic()
// Support function to unmarshal the template. This is used because the
// Input may be a TPMT_TEMPLATE and that structure does not have the same
// size as a TPMT_PUBLIC because of the difference between the 'unique' and
// 'seed' fields.
//
// If 'derive' is not NULL, then the 'seed' field is assumed to contain
// a 'label' and 'context' that are unmarshaled into 'derive'.
TPM_RC
UnmarshalToPublic(TPMT_PUBLIC*    tOut,  // OUT: output
                  TPM2B_TEMPLATE* tIn,   // IN:
                  BOOL derivation,       // IN: indicates if this is for a derivation
                  TPMS_DERIVE* labelContext  // OUT: label and context if derivation
);

//*** ObjectSetExternal()
// Set the external attributes for an object.
void ObjectSetExternal(OBJECT* object);

#endif  // _OBJECT_SPT_FP_H_
```

6.154 /tpm/include/private/prototypes/PCR_Allocate_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PCR_Allocate  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_ALLOCATE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_ALLOCATE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PLATFORM   authHandle;
    TPML_PCR_SELECTION pcrAllocation;
} PCR_Allocate_In;

// Output structure definition
typedef struct
{
    TPMI_YES_NO allocationSuccess;
    UINT32      maxPCR;
    UINT32      sizeNeeded;
    UINT32      sizeAvailable;
} PCR_Allocate_Out;

// Response code modifiers
#    define RC_PCR_Allocate_authHandle    (TPM_RC_H + TPM_RC_1)
#    define RC_PCR_Allocate_pcrAllocation (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PCR_Allocate(PCR_Allocate_In* in, PCR_Allocate_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_ALLOCATE_FP_H_
#endif    // CC_PCR_Allocate
```

6.155 /tpm/include/private/prototypes/PCR_Event_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PCR_Event  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_EVENT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_EVENT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_PCR pcrHandle;
    TPM2B_EVENT eventData;
} PCR_Event_In;

// Output structure definition
typedef struct
{
    TPML_DIGEST_VALUES digests;
} PCR_Event_Out;

// Response code modifiers
#    define RC_PCR_Event_pcrHandle (TPM_RC_H + TPM_RC_1)
#    define RC_PCR_Event_eventData (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PCR_Event(PCR_Event_In* in, PCR_Event_Out* out);
```

```
#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_EVENT_FP_H_
#endif    // CC_PCR_Event
```

6.156 /tpm/include/private/prototypes/PCR_Extend_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PCR_Extend  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_EXTEND_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_EXTEND_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_PCR        pcrHandle;
    TPML_DIGEST_VALUES digests;
} PCR_Extend_In;

// Response code modifiers
#    define RC_PCR_Extend_pcrHandle (TPM_RC_H + TPM_RC_1)
#    define RC_PCR_Extend_digests   (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PCR_Extend(PCR_Extend_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_EXTEND_FP_H_
#endif    // CC_PCR_Extend
```

6.157 /tpm/include/private/prototypes/PCR_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _PCR_FP_H_
#define _PCR_FP_H_

//*** PCRBelongsAuthGroup()
// This function indicates if a PCR belongs to a group that requires an authValue
// in order to modify the PCR.  If it does, 'groupIndex' is set to value of
// the group index.  This feature of PCR is decided by the platform specification.
//
//  Return Type: BOOL
//      TRUE(1)         PCR belongs an authorization group
//      FALSE(0)        PCR does not belong an authorization group
BOOL PCRBelongsAuthGroup(TPMI_DH_PCR handle,     // IN: handle of PCR
                         UINT32*     groupIndex  // OUT: group index if PCR belongs a
                         //      group that allows authValue.  If PCR
                         //      does not belong to an authorization
                         //      group, the value in this parameter is
                         //      invalid
);

//*** PCRBelongsPolicyGroup()
// This function indicates if a PCR belongs to a group that requires a policy
// authorization in order to modify the PCR.  If it does, 'groupIndex' is set
// to value of the group index.  This feature of PCR is decided by the platform
// specification.
//
//  Return Type: BOOL
```

```
//      TRUE(1)         PCR belongs to a policy group
//      FALSE(0)        PCR does not belong to a policy group
BOOL PCRBelongsPolicyGroup(
    TPMI_DH_PCR handle,     // IN: handle of PCR
    UINT32*     groupIndex  // OUT: group index if PCR belongs a group that
                            //     allows policy.  If PCR does not belong to
                            //     a policy group, the value in this
                            //     parameter is invalid
);

//*** PCRPolicyIsAvailable()
// This function indicates if a policy is available for a PCR.
//
//  Return Type: BOOL
//      TRUE(1)         the PCR may be authorized by policy
//      FALSE(0)        the PCR does not allow policy
BOOL PCRPolicyIsAvailable(TPMI_DH_PCR handle  // IN: PCR handle
);

//*** PCRGetAuthValue()
// This function is used to access the authValue of a PCR.  If PCR does not
// belong to an authValue group, an EmptyAuth will be returned.
TPM2B_AUTH* PCRGetAuthValue(TPMI_DH_PCR handle  // IN: PCR handle
);

//*** PCRGetAuthPolicy()
// This function is used to access the authorization policy of a PCR. It sets
// 'policy' to the authorization policy and returns the hash algorithm for policy
//  If the PCR does not allow a policy, TPM_ALG_NULL is returned.
TPMI_ALG_HASH
PCRGetAuthPolicy(TPMI_DH_PCR   handle,  // IN: PCR handle
                 TPM2B_DIGEST* policy   // OUT: policy of PCR
);

//*** PCRManufacture()
// This function is used to initialize the policies when a TPM is manufactured.
// This function would only be called in a manufacturing environment or in
// a TPM simulator.
void PCRManufacture(void);

//*** PcrIsAllocated()
// This function indicates if a PCR number for the particular hash algorithm
// is allocated.
//  Return Type: BOOL
//      TRUE(1)         PCR is allocated
//      FALSE(0)        PCR is not allocated
BOOL PcrIsAllocated(UINT32        pcr,     // IN: The number of the PCR
                    TPMI_ALG_HASH hashAlg  // IN: The PCR algorithm
);

//*** PcrDrtm()
// This function does the DRTM and H-CRTM processing it is called from
// _TPM_Hash_End.
void PcrDrtm(const TPMI_DH_PCR pcrHandle,  // IN: the index of the PCR to be
                                           //     modified
             const TPMI_ALG_HASH hash,     // IN: the bank identifier
             const TPM2B_DIGEST* digest    // IN: the digest to modify the PCR
);

//*** PCR_ClearAuth()
// This function is used to reset the PCR authorization values. It is called
// on TPM2_Startup(CLEAR) and TPM2_Clear().
void PCR_ClearAuth(void);

//*** PCRStartup()
// This function initializes the PCR subsystem at TPM2_Startup().
```

```
BOOL PCRStartup(STARTUP_TYPE type,     // IN: startup type
                BYTE         locality  // IN: startup locality
);

//*** PCRStateSave()
// This function is used to save the PCR values that will be restored on TPM Resume.
void PCRStateSave(TPM_SU type  // IN: startup type
);

//*** PCRIsStateSaved()
// This function indicates if the selected PCR is a PCR that is state saved
// on TPM2_Shutdown(STATE). The return value is based on PCR attributes.
//  Return Type: BOOL
//      TRUE(1)         PCR is state saved
//      FALSE(0)        PCR is not state saved
BOOL PCRIsStateSaved(TPMI_DH_PCR handle  // IN: PCR handle to be extended
);

//*** PCRIsResetAllowed()
// This function indicates if a PCR may be reset by the current command locality.
// The return value is based on PCR attributes, and not the PCR allocation.
//  Return Type: BOOL
//      TRUE(1)         TPM2_PCR_Reset is allowed
//      FALSE(0)        TPM2_PCR_Reset is not allowed
BOOL PCRIsResetAllowed(TPMI_DH_PCR handle  // IN: PCR handle to be extended
);

//*** PCRChanged()
// This function checks a PCR handle to see if the attributes for the PCR are set
// so that any change to the PCR causes an increment of the pcrCounter. If it does,
// then the function increments the counter. Will also bump the counter if the
// handle is zero which means that PCR 0 can not be in the TCB group. Bump on zero
// is used by TPM2_Clear().
void PCRChanged(TPM_HANDLE pcrHandle  // IN: the handle of the PCR that changed.
);

//*** PCRIsExtendAllowed()
// This function indicates a PCR may be extended at the current command locality.
// The return value is based on PCR attributes, and not the PCR allocation.
//  Return Type: BOOL
//      TRUE(1)         extend is allowed
//      FALSE(0)        extend is not allowed
BOOL PCRIsExtendAllowed(TPMI_DH_PCR handle  // IN: PCR handle to be extended
);

//*** PCRExtend()
// This function is used to extend a PCR in a specific bank.
void PCRExtend(TPMI_DH_PCR   handle,  // IN: PCR handle to be extended
               TPMI_ALG_HASH hash,    // IN: hash algorithm of PCR
               UINT32        size,    // IN: size of data to be extended
               BYTE*         data     // IN: data to be extended
);

//*** PCRComputeCurrentDigest()
// This function computes the digest of the selected PCR.
//
// As a side-effect, 'selection' is modified so that only the implemented PCR
// will have their bits still set.
void PCRComputeCurrentDigest(
    TPMI_ALG_HASH       hashAlg,    // IN: hash algorithm to compute digest
    TPML_PCR_SELECTION* selection,  // IN/OUT: PCR selection (filtered on
                                    //     output)
    TPM2B_DIGEST* digest            // OUT: digest
);

//*** PCRRead()
```

```
// This function is used to read a list of selected PCR.  If the requested PCR
// number exceeds the maximum number that can be output, the 'selection' is
// adjusted to reflect the actual output PCR.
void PCRRead(TPML_PCR_SELECTION* selection,  // IN/OUT: PCR selection (filtered on
                                             //     output)
             TPML_DIGEST* digest,            // OUT: digest
             UINT32*      pcrCounter  // OUT: the current value of PCR generation
                                      //     number
);

//*** PCRAllocate()
// This function is used to change the PCR allocation.
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        allocate failed
//      TPM_RC_PCR              improper allocation
TPM_RC
PCRAllocate(TPML_PCR_SELECTION* allocate,      // IN: required allocation
            UINT32*             maxPCR,        // OUT: Maximum number of PCR
            UINT32*             sizeNeeded,    // OUT: required space
            UINT32*             sizeAvailable  // OUT: available space
);

//*** PCRSetValue()
// This function is used to set the designated PCR in all banks to an initial value.
// The initial value is signed and will be sign extended into the entire PCR.
//
void PCRSetValue(TPM_HANDLE handle,       // IN: the handle of the PCR to set
                 INT8       initialValue  // IN: the value to set
);

//*** PCRResetDynamics
// This function is used to reset a dynamic PCR to 0.  This function is used in
// DRTM sequence.
void PCRResetDynamics(void);

//*** PCRCapGetAllocation()
// This function is used to get the current allocation of PCR banks.
//  Return Type: TPMI_YES_NO
//      YES         if the return count is 0
//      NO          if the return count is not 0
TPMI_YES_NO
PCRCapGetAllocation(UINT32              count,        // IN: count of return
                    TPML_PCR_SELECTION* pcrSelection  // OUT: PCR allocation list
);

//*** PCRCapGetProperties()
// This function returns a list of PCR properties starting at 'property'.
//  Return Type: TPMI_YES_NO
//      YES         if no more property is available
//      NO          if there are more properties not reported
TPMI_YES_NO
PCRCapGetProperties(TPM_PT_PCR property,  // IN: the starting PCR property
                    UINT32     count,     // IN: count of returned properties
                    TPML_TAGGED_PCR_PROPERTY* select  // OUT: PCR select
);

//*** PCRGetProperty()
// This function returns the selected PCR property.
//  Return Type: BOOL
//      TRUE(1)         the property type is implemented
//      FALSE(0)        the property type is not implemented
BOOL PCRGetProperty(TPM_PT_PCR property, TPMS_TAGGED_PCR_SELECT* select);

//*** PCRCapGetHandles()
// This function is used to get a list of handles of PCR, started from 'handle'.
// If 'handle' exceeds the maximum PCR handle range, an empty list will be
```

```
// returned and the return value will be NO.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
PCRCapGetHandles(TPMI_DH_PCR  handle,     // IN: start handle
                 UINT32       count,      // IN: count of returned handles
                 TPML_HANDLE* handleList  // OUT: list of handle
);

//*** PCRCapGetOneHandle()
// This function is used to check whether a PCR handle exists.
BOOL PCRCapGetOneHandle(TPMI_DH_PCR handle  // IN: handle
);

#endif  // _PCR_FP_H_
```

6.158 /tpm/include/private/prototypes/PCR_Read_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PCR_Read  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_READ_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_READ_FP_H_

// Input structure definition
typedef struct
{
    TPML_PCR_SELECTION pcrSelectionIn;
} PCR_Read_In;

// Output structure definition
typedef struct
{
    UINT32             pcrUpdateCounter;
    TPML_PCR_SELECTION pcrSelectionOut;
    TPML_DIGEST        pcrValues;
} PCR_Read_Out;

// Response code modifiers
#    define RC_PCR_Read_pcrSelectionIn (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PCR_Read(PCR_Read_In* in, PCR_Read_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_READ_FP_H_
#endif    // CC_PCR_Read
```

6.159 /tpm/include/private/prototypes/PCR_Reset_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PCR_Reset  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_RESET_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_RESET_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_PCR pcrHandle;
} PCR_Reset_In;
```

```
// Response code modifiers
#    define RC_PCR_Reset_pcrHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PCR_Reset(PCR_Reset_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_RESET_FP_H_
#endif    // CC_PCR_Reset
```

6.160 /tpm/include/private/prototypes/PCR_SetAuthPolicy_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PCR_SetAuthPolicy  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_SETAUTHPOLICY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_SETAUTHPOLICY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PLATFORM authHandle;
    TPM2B_DIGEST     authPolicy;
    TPMI_ALG_HASH    hashAlg;
    TPMI_DH_PCR      pcrNum;
} PCR_SetAuthPolicy_In;

// Response code modifiers
#    define RC_PCR_SetAuthPolicy_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_PCR_SetAuthPolicy_authPolicy (TPM_RC_P + TPM_RC_1)
#    define RC_PCR_SetAuthPolicy_hashAlg    (TPM_RC_P + TPM_RC_2)
#    define RC_PCR_SetAuthPolicy_pcrNum     (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_PCR_SetAuthPolicy(PCR_SetAuthPolicy_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_SETAUTHPOLICY_FP_H_
#endif    // CC_PCR_SetAuthPolicy
```

6.161 /tpm/include/private/prototypes/PCR_SetAuthValue_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PCR_SetAuthValue  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_SETAUTHVALUE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_SETAUTHVALUE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_PCR  pcrHandle;
    TPM2B_DIGEST auth;
} PCR_SetAuthValue_In;

// Response code modifiers
#    define RC_PCR_SetAuthValue_pcrHandle (TPM_RC_H + TPM_RC_1)
#    define RC_PCR_SetAuthValue_auth      (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
```

```
TPM2_PCR_SetAuthValue(PCR_SetAuthValue_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PCR_SETAUTHVALUE_FP_H_
#endif    // CC_PCR_SetAuthValue
```

6.162 /tpm/include/private/prototypes/PolicyAuthorizeNV_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyAuthorizeNV  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHORIZENV_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHORIZENV_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_SH_POLICY   policySession;
} PolicyAuthorizeNV_In;

// Response code modifiers
#    define RC_PolicyAuthorizeNV_authHandle    (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyAuthorizeNV_nvIndex       (TPM_RC_H + TPM_RC_2)
#    define RC_PolicyAuthorizeNV_policySession (TPM_RC_H + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_PolicyAuthorizeNV(PolicyAuthorizeNV_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHORIZENV_FP_H_
#endif    // CC_PolicyAuthorizeNV
```

6.163 /tpm/include/private/prototypes/PolicyAuthorize_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyAuthorize  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHORIZE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHORIZE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY   policySession;
    TPM2B_DIGEST     approvedPolicy;
    TPM2B_NONCE      policyRef;
    TPM2B_NAME       keySign;
    TPMT_TK_VERIFIED checkTicket;
} PolicyAuthorize_In;

// Response code modifiers
#    define RC_PolicyAuthorize_policySession  (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyAuthorize_approvedPolicy (TPM_RC_P + TPM_RC_1)
#    define RC_PolicyAuthorize_policyRef      (TPM_RC_P + TPM_RC_2)
#    define RC_PolicyAuthorize_keySign        (TPM_RC_P + TPM_RC_3)
#    define RC_PolicyAuthorize_checkTicket    (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_PolicyAuthorize(PolicyAuthorize_In* in);
```

```
#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHORIZE_FP_H_
#endif    // CC_PolicyAuthorize
```

6.164 /tpm/include/private/prototypes/PolicyAuthValue_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyAuthValue  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHVALUE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHVALUE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
} PolicyAuthValue_In;

// Response code modifiers
#    define RC_PolicyAuthValue_policySession (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyAuthValue(PolicyAuthValue_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYAUTHVALUE_FP_H_
#endif    // CC_PolicyAuthValue
```

6.165 /tpm/include/private/prototypes/PolicyCapability_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyCapability  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCAPABILITY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCAPABILITY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_OPERAND  operandB;
    UINT16         offset;
    TPM_EO         operation;
    TPM_CAP        capability;
    UINT32         property;
} PolicyCapability_In;

// Response code modifiers
#    define RC_PolicyCapability_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyCapability_operandB      (TPM_RC_P + TPM_RC_1)
#    define RC_PolicyCapability_offset        (TPM_RC_P + TPM_RC_2)
#    define RC_PolicyCapability_operation     (TPM_RC_P + TPM_RC_3)
#    define RC_PolicyCapability_capability    (TPM_RC_P + TPM_RC_4)
#    define RC_PolicyCapability_property      (TPM_RC_P + TPM_RC_5)

// Function prototype
TPM_RC
TPM2_PolicyCapability(PolicyCapability_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCAPABILITY_FP_H_
#endif    // CC_PolicyCapability
```

6.166 /tpm/include/private/prototypes/PolicyCommandCode_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyCommandCode  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCOMMANDCODE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCOMMANDCODE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM_CC         code;
} PolicyCommandCode_In;

// Response code modifiers
#    define RC_PolicyCommandCode_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyCommandCode_code          (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyCommandCode(PolicyCommandCode_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCOMMANDCODE_FP_H_
#endif    // CC_PolicyCommandCode
```

6.167 /tpm/include/private/prototypes/PolicyCounterTimer_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyCounterTimer  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCOUNTERTIMER_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCOUNTERTIMER_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_OPERAND  operandB;
    UINT16         offset;
    TPM_EO         operation;
} PolicyCounterTimer_In;

// Response code modifiers
#    define RC_PolicyCounterTimer_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyCounterTimer_operandB      (TPM_RC_P + TPM_RC_1)
#    define RC_PolicyCounterTimer_offset        (TPM_RC_P + TPM_RC_2)
#    define RC_PolicyCounterTimer_operation     (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_PolicyCounterTimer(PolicyCounterTimer_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCOUNTERTIMER_FP_H_
#endif    // CC_PolicyCounterTimer
```

6.168 /tpm/include/private/prototypes/PolicyCpHash_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyCpHash  // Command must be enabled
```

```
#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCPHASH_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCPHASH_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST   cpHashA;
} PolicyCpHash_In;

// Response code modifiers
#    define RC_PolicyCpHash_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyCpHash_cpHashA       (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyCpHash(PolicyCpHash_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYCPHASH_FP_H_
#endif    // CC_PolicyCpHash
```

6.169 /tpm/include/private/prototypes/PolicyDuplicationSelect_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyDuplicationSelect  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYDUPLICATIONSELECT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYDUPLICATIONSELECT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_NAME     objectName;
    TPM2B_NAME     newParentName;
    TPMI_YES_NO    includeObject;
} PolicyDuplicationSelect_In;

// Response code modifiers
#    define RC_PolicyDuplicationSelect_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyDuplicationSelect_objectName    (TPM_RC_P + TPM_RC_1)
#    define RC_PolicyDuplicationSelect_newParentName (TPM_RC_P + TPM_RC_2)
#    define RC_PolicyDuplicationSelect_includeObject (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_PolicyDuplicationSelect(PolicyDuplicationSelect_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYDUPLICATIONSELECT_FP_H_
#endif    // CC_PolicyDuplicationSelect
```

6.170 /tpm/include/private/prototypes/PolicyGetDigest_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyGetDigest  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYGETDIGEST_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYGETDIGEST_FP_H_

// Input structure definition
typedef struct
```

```
{
    TPMI_SH_POLICY policySession;
} PolicyGetDigest_In;

// Output structure definition
typedef struct
{
    TPM2B_DIGEST policyDigest;
} PolicyGetDigest_Out;

// Response code modifiers
#    define RC_PolicyGetDigest_policySession (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyGetDigest(PolicyGetDigest_In* in, PolicyGetDigest_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYGETDIGEST_FP_H_
#endif    // CC_PolicyGetDigest
```

6.171 /tpm/include/private/prototypes/PolicyLocality_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyLocality  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYLOCALITY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYLOCALITY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPMA_LOCALITY  locality;
} PolicyLocality_In;

// Response code modifiers
#    define RC_PolicyLocality_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyLocality_locality      (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyLocality(PolicyLocality_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYLOCALITY_FP_H_
#endif    // CC_PolicyLocality
```

6.172 /tpm/include/private/prototypes/PolicyNameHash_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyNameHash  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNAMEHASH_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNAMEHASH_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST   nameHash;
} PolicyNameHash_In;

// Response code modifiers
```

```
#    define RC_PolicyNameHash_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyNameHash_nameHash      (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyNameHash(PolicyNameHash_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNAMEHASH_FP_H_
#endif    // CC_PolicyNameHash
```

6.173 /tpm/include/private/prototypes/PolicyNvWritten_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyNvWritten  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNVWRITTEN_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNVWRITTEN_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPMI_YES_NO    writtenSet;
} PolicyNvWritten_In;

// Response code modifiers
#    define RC_PolicyNvWritten_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyNvWritten_writtenSet    (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyNvWritten(PolicyNvWritten_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNVWRITTEN_FP_H_
#endif    // CC_PolicyNvWritten
```

6.174 /tpm/include/private/prototypes/PolicyNV_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyNV  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNV_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNV_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_NV_AUTH  authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_SH_POLICY   policySession;
    TPM2B_OPERAND    operandB;
    UINT16           offset;
    TPM_EO           operation;
} PolicyNV_In;

// Response code modifiers
#    define RC_PolicyNV_authHandle    (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyNV_nvIndex       (TPM_RC_H + TPM_RC_2)
#    define RC_PolicyNV_policySession (TPM_RC_H + TPM_RC_3)
#    define RC_PolicyNV_operandB      (TPM_RC_P + TPM_RC_1)
#    define RC_PolicyNV_offset        (TPM_RC_P + TPM_RC_2)
#    define RC_PolicyNV_operation     (TPM_RC_P + TPM_RC_3)
```

```
// Function prototype
TPM_RC
TPM2_PolicyNV(PolicyNV_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYNV_FP_H_
#endif    // CC_PolicyNV
```

6.175 /tpm/include/private/prototypes/PolicyOR_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyOR  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYOR_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYOR_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPML_DIGEST    pHashList;
} PolicyOR_In;

// Response code modifiers
#    define RC_PolicyOR_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyOR_pHashList     (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyOR(PolicyOR_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYOR_FP_H_
#endif    // CC_PolicyOR
```

6.176 /tpm/include/private/prototypes/PolicyParameters_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyParameters  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPARAMETERS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPARAMETERS_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST   pHash;
} PolicyParameters_In;

// Response code modifiers
#    define RC_PolicyParameters_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyParameters_pHash         (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyParameters(PolicyParameters_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPARAMETERS_FP_H_
#endif    // CC_PolicyParameters
```

6.177 /tpm/include/private/prototypes/PolicyPassword_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyPassword  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPASSWORD_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPASSWORD_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
} PolicyPassword_In;

// Response code modifiers
#    define RC_PolicyPassword_policySession (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyPassword(PolicyPassword_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPASSWORD_FP_H_
#endif    // CC_PolicyPassword
```

6.178 /tpm/include/private/prototypes/PolicyPCR_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyPCR  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPCR_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPCR_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY     policySession;
    TPM2B_DIGEST       pcrDigest;
    TPML_PCR_SELECTION pcrs;
} PolicyPCR_In;

// Response code modifiers
#    define RC_PolicyPCR_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyPCR_pcrDigest     (TPM_RC_P + TPM_RC_1)
#    define RC_PolicyPCR_pcrs          (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_PolicyPCR(PolicyPCR_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPCR_FP_H_
#endif    // CC_PolicyPCR
```

6.179 /tpm/include/private/prototypes/PolicyPhysicalPresence_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyPhysicalPresence  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPHYSICALPRESENCE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPHYSICALPRESENCE_FP_H_
```

```
// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
} PolicyPhysicalPresence_In;

// Response code modifiers
#    define RC_PolicyPhysicalPresence_policySession (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyPhysicalPresence(PolicyPhysicalPresence_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYPHYSICALPRESENCE_FP_H_
#endif    // CC_PolicyPhysicalPresence
```

6.180 /tpm/include/private/prototypes/PolicyRestart_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyRestart  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYRESTART_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYRESTART_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY sessionHandle;
} PolicyRestart_In;

// Response code modifiers
#    define RC_PolicyRestart_sessionHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyRestart(PolicyRestart_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYRESTART_FP_H_
#endif    // CC_PolicyRestart
```

6.181 /tpm/include/private/prototypes/PolicySecret_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicySecret  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYSECRET_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYSECRET_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_ENTITY authHandle;
    TPMI_SH_POLICY policySession;
    TPM2B_NONCE    nonceTPM;
    TPM2B_DIGEST   cpHashA;
    TPM2B_NONCE    policyRef;
    INT32          expiration;
} PolicySecret_In;

// Output structure definition
typedef struct
{
```

```
    TPM2B_TIMEOUT timeout;
    TPMT_TK_AUTH  policyTicket;
} PolicySecret_Out;

// Response code modifiers
#    define RC_PolicySecret_authHandle    (TPM_RC_H + TPM_RC_1)
#    define RC_PolicySecret_policySession (TPM_RC_H + TPM_RC_2)
#    define RC_PolicySecret_nonceTPM      (TPM_RC_P + TPM_RC_1)
#    define RC_PolicySecret_cpHashA       (TPM_RC_P + TPM_RC_2)
#    define RC_PolicySecret_policyRef     (TPM_RC_P + TPM_RC_3)
#    define RC_PolicySecret_expiration    (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_PolicySecret(PolicySecret_In* in, PolicySecret_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYSECRET_FP_H_
#endif    // CC_PolicySecret
```

6.182 /tpm/include/private/prototypes/PolicySigned_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicySigned  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYSIGNED_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYSIGNED_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT authObject;
    TPMI_SH_POLICY policySession;
    TPM2B_NONCE    nonceTPM;
    TPM2B_DIGEST   cpHashA;
    TPM2B_NONCE    policyRef;
    INT32          expiration;
    TPMT_SIGNATURE auth;
} PolicySigned_In;

// Output structure definition
typedef struct
{
    TPM2B_TIMEOUT timeout;
    TPMT_TK_AUTH  policyTicket;
} PolicySigned_Out;

// Response code modifiers
#    define RC_PolicySigned_authObject    (TPM_RC_H + TPM_RC_1)
#    define RC_PolicySigned_policySession (TPM_RC_H + TPM_RC_2)
#    define RC_PolicySigned_nonceTPM      (TPM_RC_P + TPM_RC_1)
#    define RC_PolicySigned_cpHashA       (TPM_RC_P + TPM_RC_2)
#    define RC_PolicySigned_policyRef     (TPM_RC_P + TPM_RC_3)
#    define RC_PolicySigned_expiration    (TPM_RC_P + TPM_RC_4)
#    define RC_PolicySigned_auth          (TPM_RC_P + TPM_RC_5)

// Function prototype
TPM_RC
TPM2_PolicySigned(PolicySigned_In* in, PolicySigned_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYSIGNED_FP_H_
#endif    // CC_PolicySigned
```

6.183 /tpm/include/private/prototypes/PolicyTemplate_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyTemplate  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYTEMPLATE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYTEMPLATE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST   templateHash;
} PolicyTemplate_In;

// Response code modifiers
#    define RC_PolicyTemplate_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyTemplate_templateHash  (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_PolicyTemplate(PolicyTemplate_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYTEMPLATE_FP_H_
#endif    // CC_PolicyTemplate
```

6.184 /tpm/include/private/prototypes/PolicyTicket_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PolicyTicket  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYTICKET_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYTICKET_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_TIMEOUT  timeout;
    TPM2B_DIGEST   cpHashA;
    TPM2B_NONCE    policyRef;
    TPM2B_NAME     authName;
    TPMT_TK_AUTH   ticket;
} PolicyTicket_In;

// Response code modifiers
#    define RC_PolicyTicket_policySession (TPM_RC_H + TPM_RC_1)
#    define RC_PolicyTicket_timeout       (TPM_RC_P + TPM_RC_1)
#    define RC_PolicyTicket_cpHashA       (TPM_RC_P + TPM_RC_2)
#    define RC_PolicyTicket_policyRef     (TPM_RC_P + TPM_RC_3)
#    define RC_PolicyTicket_authName      (TPM_RC_P + TPM_RC_4)
#    define RC_PolicyTicket_ticket        (TPM_RC_P + TPM_RC_5)

// Function prototype
TPM_RC
TPM2_PolicyTicket(PolicyTicket_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICYTICKET_FP_H_
#endif    // CC_PolicyTicket
```

6.185 /tpm/include/private/prototypes/Policy_AC_SendSelect_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Policy_AC_SendSelect  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICY_AC_SENDSELECT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICY_AC_SENDSELECT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_SH_POLICY policySession;
    TPM2B_NAME     objectName;
    TPM2B_NAME     authHandleName;
    TPM2B_NAME     acName;
    TPMI_YES_NO    includeObject;
} Policy_AC_SendSelect_In;

// Response code modifiers
#    define RC_Policy_AC_SendSelect_policySession  (TPM_RC_H + TPM_RC_1)
#    define RC_Policy_AC_SendSelect_objectName     (TPM_RC_P + TPM_RC_1)
#    define RC_Policy_AC_SendSelect_authHandleName (TPM_RC_P + TPM_RC_2)
#    define RC_Policy_AC_SendSelect_acName         (TPM_RC_P + TPM_RC_3)
#    define RC_Policy_AC_SendSelect_includeObject  (TPM_RC_P + TPM_RC_4)

// Function prototype
TPM_RC
TPM2_Policy_AC_SendSelect(Policy_AC_SendSelect_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_POLICY_AC_SENDSELECT_FP_H_
#endif    // CC_Policy_AC_SendSelect
```

6.186 /tpm/include/private/prototypes/Policy_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _POLICY_SPT_FP_H_
#define _POLICY_SPT_FP_H_

//** Functions
//*** PolicyParameterChecks()
// This function validates the common parameters of TPM2_PolicySiged()
// and TPM2_PolicySecret(). The common parameters are 'nonceTPM',
// 'expiration', and 'cpHashA'.
TPM_RC
PolicyParameterChecks(SESSION*      session,
                      UINT64        authTimeout,
                      TPM2B_DIGEST* cpHashA,
                      TPM2B_NONCE*  nonce,
                      TPM_RC        blameNonce,
                      TPM_RC        blameCpHash,
                      TPM_RC        blameExpiration);

//*** PolicyContextUpdate()
// Update policy hash
//      Update the policyDigest in policy session by extending policyRef and
//      objectName to it. This will also update the cpHash if it is present.
//
//  Return Type: void
void PolicyContextUpdate(
    TPM_CC        commandCode,    // IN: command code
```

```
    TPM2B_NAME*    name,           // IN: name of entity
    TPM2B_NONCE*   ref,            // IN: the reference data
    TPM2B_DIGEST*  cpHash,         // IN: the cpHash (optional)
    UINT64         policyTimeout,  // IN: the timeout value for the policy
    SESSION*       session         // IN/OUT: policy session to be updated
);

//*** ComputeAuthTimeout()
// This function is used to determine what the authorization timeout value for
// the session should be.
UINT64
ComputeAuthTimeout(SESSION* session,   // IN: the session containing the time
                                       //     values
                   INT32 expiration,   // IN: either the number of seconds from
                                       //     the start of the session or the
                                       //     time in g_timer;
                   TPM2B_NONCE* nonce  // IN: indicator of the time base
);

//*** PolicyDigestClear()
// Function to reset the policyDigest of a session
void PolicyDigestClear(SESSION* session);

//*** PolicySptCheckCondition()
// Checks to see if the condition in the policy is satisfied.
BOOL PolicySptCheckCondition(TPM_EO operation, BYTE* opA, BYTE* opB, UINT16 size);

#endif  // _POLICY_SPT_FP_H_
```

6.187 /tpm/include/private/prototypes/Power_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 11:00:49AM
 */

#ifndef _POWER_FP_H_
#define _POWER_FP_H_

//*** TPMInit()
// This function is used to process a power on event.
void TPMInit(void);

//*** TPMRegisterStartup()
// This function registers the fact that the TPM has been initialized
// (a TPM2_Startup() has completed successfully).
BOOL TPMRegisterStartup(void);

//*** TPMIsStarted()
// Indicates if the TPM has been initialized (a TPM2_Startup() has completed
// successfully after a _TPM_Init).
//  Return Type: BOOL
//      TRUE(1)         TPM has been initialized
//      FALSE(0)        TPM has not been initialized
BOOL TPMIsStarted(void);

#endif  // _POWER_FP_H_
```

6.188 /tpm/include/private/prototypes/PP_Commands_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_PP_Commands  // Command must be enabled
```

```
#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_PP_COMMANDS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_PP_COMMANDS_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PLATFORM auth;
    TPML_CC          setList;
    TPML_CC          clearList;
} PP_Commands_In;

// Response code modifiers
#    define RC_PP_Commands_auth      (TPM_RC_H + TPM_RC_1)
#    define RC_PP_Commands_setList   (TPM_RC_P + TPM_RC_1)
#    define RC_PP_Commands_clearList (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_PP_Commands(PP_Commands_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_PP_COMMANDS_FP_H_
#endif    // CC_PP_Commands
```

6.189 /tpm/include/private/prototypes/PP_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _PP_FP_H_
#define _PP_FP_H_

//*** PhysicalPresencePreInstall_Init()
// This function is used to initialize the array of commands that always require
// confirmation with physical presence. The array is an array of bits that
// has a correspondence with the command code.
//
// This command should only ever be executable in a manufacturing setting or in
// a simulation.
//
// When set, these cannot be cleared.
//
void PhysicalPresencePreInstall_Init(void);

//*** PhysicalPresenceCommandSet()
// This function is used to set the indicator that a command requires
// PP confirmation.
void PhysicalPresenceCommandSet(TPM_CC commandCode  // IN: command code
);

//*** PhysicalPresenceCommandClear()
// This function is used to clear the indicator that a command requires PP
// confirmation.
void PhysicalPresenceCommandClear(TPM_CC commandCode  // IN: command code
);

//*** PhysicalPresenceIsRequired()
// This function indicates if PP confirmation is required for a command.
//  Return Type: BOOL
//      TRUE(1)         physical presence is required
//      FALSE(0)        physical presence is not required
BOOL PhysicalPresenceIsRequired(COMMAND_INDEX commandIndex  // IN: command index
);
```

```
//*** PhysicalPresenceCapGetCCList()
// This function returns a list of commands that require PP confirmation. The
// list starts from the first implemented command that has a command code that
// the same or greater than 'commandCode'.
//  Return Type: TPMI_YES_NO
//      YES         if there are more command codes available
//      NO          all the available command codes have been returned
TPMI_YES_NO
PhysicalPresenceCapGetCCList(TPM_CC   commandCode,  // IN: start command code
                             UINT32   count,        // IN: count of returned TPM_CC
                             TPML_CC* commandList   // OUT: list of TPM_CC
);

//*** PhysicalPresenceCapGetOneCC()
// This function returns true if the command requires Physical Presence.
BOOL PhysicalPresenceCapGetOneCC(TPM_CC commandCode  // IN: command code
);

#endif  // _PP_FP_H_
```

6.190 /tpm/include/private/prototypes/PropertyCap_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _PROPERTY_CAP_FP_H_
#define _PROPERTY_CAP_FP_H_

//*** TPMCapGetProperties()
// This function is used to get the TPM_PT values. The search of properties will
// start at 'property' and continue until 'propertyList' has as many values as
// will fit, or the last property has been reported, or the list has as many
// values as requested in 'count'.
//  Return Type: TPMI_YES_NO
//  YES         more properties are available
//  NO          no more properties to be reported
TPMI_YES_NO
TPMCapGetProperties(TPM_PT property,  // IN: the starting TPM property
                    UINT32 count,     // IN: maximum number of returned
                                      //     properties
                    TPML_TAGGED_TPM_PROPERTY* propertyList  // OUT: property list
);

//*** TPMCapGetOneProperty()
// This function returns a single TPM property, if present.
BOOL TPMCapGetOneProperty(TPM_PT                 pt,       // IN: the TPM property
                          TPMS_TAGGED_PROPERTY* property  // OUT: tagged property
);

#endif  // _PROPERTY_CAP_FP_H_
```

6.191 /tpm/include/private/prototypes/Quote_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Quote  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_QUOTE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_QUOTE_FP_H_

// Input structure definition
typedef struct
```

```
{
    TPMI_DH_OBJECT     signHandle;
    TPM2B_DATA         qualifyingData;
    TPMT_SIG_SCHEME    inScheme;
    TPML_PCR_SELECTION PCRselect;
} Quote_In;

// Output structure definition
typedef struct
{
    TPM2B_ATTEST   quoted;
    TPMT_SIGNATURE signature;
} Quote_Out;

// Response code modifiers
#    define RC_Quote_signHandle     (TPM_RC_H + TPM_RC_1)
#    define RC_Quote_qualifyingData (TPM_RC_P + TPM_RC_1)
#    define RC_Quote_inScheme       (TPM_RC_P + TPM_RC_2)
#    define RC_Quote_PCRselect      (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_Quote(Quote_In* in, Quote_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_QUOTE_FP_H_
#endif    // CC_Quote
```

6.192 /tpm/include/private/prototypes/ReadClock_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ReadClock  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_READCLOCK_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_READCLOCK_FP_H_

// Output structure definition
typedef struct
{
    TPMS_TIME_INFO currentTime;
} ReadClock_Out;

// Function prototype
TPM_RC
TPM2_ReadClock(ReadClock_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_READCLOCK_FP_H_
#endif    // CC_ReadClock
```

6.193 /tpm/include/private/prototypes/ReadPublic_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ReadPublic  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_READPUBLIC_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_READPUBLIC_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT objectHandle;
} ReadPublic_In;
```

```
// Output structure definition
typedef struct
{
    TPM2B_PUBLIC outPublic;
    TPM2B_NAME   name;
    TPM2B_NAME   qualifiedName;
} ReadPublic_Out;

// Response code modifiers
#    define RC_ReadPublic_objectHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_ReadPublic(ReadPublic_In* in, ReadPublic_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_READPUBLIC_FP_H_
#endif    // CC_ReadPublic
```

6.194 /tpm/include/private/prototypes/ResponseCodeProcessing_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _RESPONSE_CODE_PROCESSING_FP_H_
#define _RESPONSE_CODE_PROCESSING_FP_H_

//** RcSafeAddToResult()
// Adds a modifier to a response code as long as the response code allows a modifier
// and no modifier has already been added.
TPM_RC
RcSafeAddToResult(TPM_RC responseCode, TPM_RC modifier);

#endif  // _RESPONSE_CODE_PROCESSING_FP_H_
```

6.195 /tpm/include/private/prototypes/Response_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _RESPONSE_FP_H_
#define _RESPONSE_FP_H_

//** BuildResponseHeader()
// Adds the response header to the response. It will update command->parameterSize
// to indicate the total size of the response.
void BuildResponseHeader(COMMAND* command,  // IN: main control structure
                         BYTE*    buffer,   // OUT: the output buffer
                         TPM_RC   result    // IN: the response code
);

#endif  // _RESPONSE_FP_H_
```

6.196 /tpm/include/private/prototypes/Rewrap_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Rewrap  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_REWRAP_FP_H_
```

```
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_REWRAP_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT          oldParent;
    TPMI_DH_OBJECT          newParent;
    TPM2B_PRIVATE           inDuplicate;
    TPM2B_NAME              name;
    TPM2B_ENCRYPTED_SECRET inSymSeed;
} Rewrap_In;

// Output structure definition
typedef struct
{
    TPM2B_PRIVATE           outDuplicate;
    TPM2B_ENCRYPTED_SECRET outSymSeed;
} Rewrap_Out;

// Response code modifiers
#    define RC_Rewrap_oldParent   (TPM_RC_H + TPM_RC_1)
#    define RC_Rewrap_newParent   (TPM_RC_H + TPM_RC_2)
#    define RC_Rewrap_inDuplicate (TPM_RC_P + TPM_RC_1)
#    define RC_Rewrap_name        (TPM_RC_P + TPM_RC_2)
#    define RC_Rewrap_inSymSeed   (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_Rewrap(Rewrap_In* in, Rewrap_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_REWRAP_FP_H_
#endif    // CC_Rewrap
```

6.197 /tpm/include/private/prototypes/RsaKeyCache_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */

#ifndef _RSA_KEY_CACHE_FP_H_
#define _RSA_KEY_CACHE_FP_H_

#if USE_RSA_KEY_CACHE

//*** RsaKeyCacheControl()
// Used to enable and disable the RSA key cache.
LIB_EXPORT void RsaKeyCacheControl(int state);

//*** GetCachedRsaKey()
//  Return Type: BOOL
//      TRUE(1)         key loaded
//      FALSE(0)        key not loaded
BOOL GetCachedRsaKey(TPMT_PUBLIC*    publicArea,
                     TPMT_SENSITIVE* sensitive,
                     RAND_STATE*     rand  // IN: if not NULL, the deterministic
                                           //     RNG state
);
#endif  // defined SIMULATION && defined USE_RSA_KEY_CACHE

#endif  // _RSA_KEY_CACHE_FP_H_
```

6.198 /tpm/include/private/prototypes/RSA_Decrypt_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_RSA_Decrypt  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_RSA_DECRYPT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_RSA_DECRYPT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT       keyHandle;
    TPM2B_PUBLIC_KEY_RSA cipherText;
    TPMT_RSA_DECRYPT     inScheme;
    TPM2B_DATA           label;
} RSA_Decrypt_In;

// Output structure definition
typedef struct
{
    TPM2B_PUBLIC_KEY_RSA message;
} RSA_Decrypt_Out;

// Response code modifiers
#    define RC_RSA_Decrypt_keyHandle  (TPM_RC_H + TPM_RC_1)
#    define RC_RSA_Decrypt_cipherText (TPM_RC_P + TPM_RC_1)
#    define RC_RSA_Decrypt_inScheme   (TPM_RC_P + TPM_RC_2)
#    define RC_RSA_Decrypt_label      (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_RSA_Decrypt(RSA_Decrypt_In* in, RSA_Decrypt_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_RSA_DECRYPT_FP_H_
#endif    // CC_RSA_Decrypt
```

6.199 /tpm/include/private/prototypes/RSA_Encrypt_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_RSA_Encrypt  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_RSA_ENCRYPT_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_RSA_ENCRYPT_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT       keyHandle;
    TPM2B_PUBLIC_KEY_RSA message;
    TPMT_RSA_DECRYPT     inScheme;
    TPM2B_DATA           label;
} RSA_Encrypt_In;

// Output structure definition
typedef struct
{
    TPM2B_PUBLIC_KEY_RSA outData;
} RSA_Encrypt_Out;

// Response code modifiers
#    define RC_RSA_Encrypt_keyHandle (TPM_RC_H + TPM_RC_1)
#    define RC_RSA_Encrypt_message   (TPM_RC_P + TPM_RC_1)
#    define RC_RSA_Encrypt_inScheme  (TPM_RC_P + TPM_RC_2)
```

```
#    define RC_RSA_Encrypt_label     (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_RSA_Encrypt(RSA_Encrypt_In* in, RSA_Encrypt_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_RSA_ENCRYPT_FP_H_
#endif    // CC_RSA_Encrypt
```

6.200 /tpm/include/private/prototypes/SelfTest_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_SelfTest  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SELFTEST_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SELFTEST_FP_H_

// Input structure definition
typedef struct
{
    TPMI_YES_NO fullTest;
} SelfTest_In;

// Response code modifiers
#    define RC_SelfTest_fullTest (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_SelfTest(SelfTest_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SELFTEST_FP_H_
#endif    // CC_SelfTest
```

6.201 /tpm/include/private/prototypes/SequenceComplete_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_SequenceComplete  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SEQUENCECOMPLETE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SEQUENCECOMPLETE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT    sequenceHandle;
    TPM2B_MAX_BUFFER  buffer;
    TPMI_RH_HIERARCHY hierarchy;
} SequenceComplete_In;

// Output structure definition
typedef struct
{
    TPM2B_DIGEST      result;
    TPMT_TK_HASHCHECK validation;
} SequenceComplete_Out;

// Response code modifiers
#    define RC_SequenceComplete_sequenceHandle (TPM_RC_H + TPM_RC_1)
#    define RC_SequenceComplete_buffer         (TPM_RC_P + TPM_RC_1)
#    define RC_SequenceComplete_hierarchy      (TPM_RC_P + TPM_RC_2)

// Function prototype
```

```
TPM_RC
TPM2_SequenceComplete(SequenceComplete_In* in, SequenceComplete_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SEQUENCECOMPLETE_FP_H_
#endif    // CC_SequenceComplete
```

6.202 /tpm/include/private/prototypes/SequenceUpdate_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_SequenceUpdate  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SEQUENCEUPDATE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SEQUENCEUPDATE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT   sequenceHandle;
    TPM2B_MAX_BUFFER buffer;
} SequenceUpdate_In;

// Response code modifiers
#    define RC_SequenceUpdate_sequenceHandle (TPM_RC_H + TPM_RC_1)
#    define RC_SequenceUpdate_buffer         (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_SequenceUpdate(SequenceUpdate_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SEQUENCEUPDATE_FP_H_
#endif    // CC_SequenceUpdate
```

6.203 /tpm/include/private/prototypes/SessionProcess_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  7, 2020  Time: 07:17:48PM
 */

#ifndef _SESSION_PROCESS_FP_H_
#define _SESSION_PROCESS_FP_H_

//*** IsDAExempted()
// This function indicates if a handle is exempted from DA logic.
// A handle is exempted if it is:
//  a) a primary seed handle;
//  b) an object with noDA bit SET;
//  c) an NV Index with TPMA_NV_NO_DA bit SET; or
//  d) a PCR handle.
//
//  Return Type: BOOL
//      TRUE(1)         handle is exempted from DA logic
//      FALSE(0)        handle is not exempted from DA logic
BOOL IsDAExempted(TPM_HANDLE handle  // IN: entity handle
);

//*** ClearCpRpHashes()
void ClearCpRpHashes(COMMAND* command);

//*** CompareNameHash()
// This function computes the name hash and compares it to the nameHash in the
// session data, returning true if they are equal.
BOOL CompareNameHash(COMMAND* command,  // IN: main parsing structure
```

```
                      SESSION* session   // IN: session structure with nameHash
);

//*** CompareParametersHash()
// This function computes the parameters hash and compares it to the pHash in
// the session data, returning true if they are equal.
BOOL CompareParametersHash(COMMAND* command,  // IN: main parsing structure
                           SESSION* session   // IN: session structure with pHash
);

//*** ParseSessionBuffer()
// This function is the entry function for command session processing.
// It iterates sessions in session area and reports if the required authorization
// has been properly provided. It also processes audit session and passes the
// information of encryption sessions to parameter encryption module.
//
//  Return Type: TPM_RC
//        various           parsing failure or authorization failure
//
TPM_RC
ParseSessionBuffer(COMMAND* command  // IN: the structure that contains
);

//*** CheckAuthNoSession()
// Function to process a command with no session associated.
// The function makes sure all the handles in the command require no authorization.
//
//  Return Type: TPM_RC
//      TPM_RC_AUTH_MISSING         failure - one or more handles require
//                                  authorization
TPM_RC
CheckAuthNoSession(COMMAND* command  // IN: command parsing structure
);

//*** BuildResponseSession()
// Function to build Session buffer in a response. The authorization data is added
// to the end of command->responseBuffer. The size of the authorization area is
// accumulated in command->authSize.
// When this is called, command->responseBuffer is pointing at the next location
// in the response buffer to be filled. This is where the authorization sessions
// will go, if any. command->parameterSize is the number of bytes that have been
// marshaled as parameters in the output buffer.
TPM_RC
BuildResponseSession(COMMAND* command  // IN: structure that has relevant command
                                       //     information
);

//*** SessionRemoveAssociationToHandle()
// This function deals with the case where an entity associated with an authorization
// is deleted during command processing. The primary use of this is to support
// UndefineSpaceSpecial().
void SessionRemoveAssociationToHandle(TPM_HANDLE handle);

#endif  // _SESSION_PROCESS_FP_H_
```

6.204 /tpm/include/private/prototypes/Session_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _SESSION_FP_H_
#define _SESSION_FP_H_
```

```
//** Startup Function -- SessionStartup()
// This function initializes the session subsystem on TPM2_Startup().
BOOL SessionStartup(STARTUP_TYPE type);

//*** SessionIsLoaded()
// This function test a session handle references a loaded session.  The handle
// must have previously been checked to make sure that it is a valid handle for
// an authorization session.
// NOTE:    A PWAP authorization does not have a session.
//
//  Return Type: BOOL
//      TRUE(1)         session is loaded
//      FALSE(0)        session is not loaded
//
BOOL SessionIsLoaded(TPM_HANDLE handle  // IN: session handle
);

//*** SessionIsSaved()
// This function test a session handle references a saved session.  The handle
// must have previously been checked to make sure that it is a valid handle for
// an authorization session.
// NOTE:    An password authorization does not have a session.
//
// This function requires that the handle be a valid session handle.
//
//  Return Type: BOOL
//      TRUE(1)         session is saved
//      FALSE(0)        session is not saved
//
BOOL SessionIsSaved(TPM_HANDLE handle  // IN: session handle
);

//*** SequenceNumberForSavedContextIsValid()
// This function validates that the sequence number and handle value within a
// saved context are valid.
BOOL SequenceNumberForSavedContextIsValid(
    TPMS_CONTEXT* context  // IN: pointer to a context structure to be
                           //     validated
);

//*** SessionPCRValueIsCurrent()
//
// This function is used to check if PCR values have been updated since the
// last time they were checked in a policy session.
//
// This function requires the session is loaded.
//  Return Type: BOOL
//      TRUE(1)         PCR value is current
//      FALSE(0)        PCR value is not current
BOOL SessionPCRValueIsCurrent(SESSION* session  // IN: session structure
);

//*** SessionGet()
// This function returns a pointer to the session object associated with a
// session handle.
//
// The function requires that the session is loaded.
SESSION* SessionGet(TPM_HANDLE handle  // IN: session handle
);

//*** SessionCreate()
//
//  This function does the detailed work for starting an authorization session.
//  This is done in a support routine rather than in the action code because
//  the session management may differ in implementations.  This implementation
//  uses a fixed memory allocation to hold sessions and a fixed allocation
```

```
//  to hold the contextID for the saved contexts.
//
//  Return Type: TPM_RC
//      TPM_RC_CONTEXT_GAP          need to recycle sessions
//      TPM_RC_SESSION_HANDLE       active session space is full
//      TPM_RC_SESSION_MEMORY       loaded session space is full
TPM_RC
SessionCreate(TPM_SE         sessionType,    // IN: the session type
              TPMI_ALG_HASH  authHash,       // IN: the hash algorithm
              TPM2B_NONCE*   nonceCaller,    // IN: initial nonceCaller
              TPMT_SYM_DEF*  symmetric,      // IN: the symmetric algorithm
              TPMI_DH_ENTITY bind,           // IN: the bind object
              TPM2B_DATA*    seed,           // IN: seed data
              TPM_HANDLE*    sessionHandle,  // OUT: the session handle
              TPM2B_NONCE*   nonceTpm        // OUT: the session nonce
);

//*** SessionContextSave()
// This function is called when a session context is to be saved.  The
// contextID of the saved session is returned.  If no contextID can be
// assigned, then the routine returns TPM_RC_CONTEXT_GAP.
// If the function completes normally, the session slot will be freed.
//
// This function requires that 'handle' references a loaded session.
// Otherwise, it should not be called at the first place.
//
//  Return Type: TPM_RC
//      TPM_RC_CONTEXT_GAP              a contextID could not be assigned
//      TPM_RC_TOO_MANY_CONTEXTS        the counter maxed out
//
TPM_RC
SessionContextSave(TPM_HANDLE       handle,    // IN: session handle
                   CONTEXT_COUNTER* contextID  // OUT: assigned contextID
);

//*** SessionContextLoad()
// This function is used to load a session from saved context.  The session
// handle must be for a saved context.
//
// If the gap is at a maximum, then the only session that can be loaded is
// the oldest session, otherwise TPM_RC_CONTEXT_GAP is returned.
//
// This function requires that 'handle' references a valid saved session.
//
//  Return Type: TPM_RC
//      TPM_RC_SESSION_MEMORY       no free session slots
//      TPM_RC_CONTEXT_GAP          the gap count is maximum and this
//                                  is not the oldest saved context
//
TPM_RC
SessionContextLoad(SESSION_BUF* session,  // IN: session structure from saved context
                   TPM_HANDLE*  handle    // IN/OUT: session handle
);

//*** SessionFlush()
// This function is used to flush a session referenced by its handle.  If the
// session associated with 'handle' is loaded, the session array entry is
// marked as available.
//
// This function requires that 'handle' be a valid active session.
//
void SessionFlush(TPM_HANDLE handle  // IN: loaded or saved session handle
);

//*** SessionComputeBoundEntity()
// This function computes the binding value for a session.  The binding value
```

```
// for a reserved handle is the handle itself.  For all the other entities,
// the authValue at the time of binding is included to prevent squatting.
// For those values, the Name and the authValue are concatenated
// into the bind buffer.  If they will not both fit, the will be overlapped
// by XORing bytes.  If XOR is required, the bind value will be full.
void SessionComputeBoundEntity(TPMI_DH_ENTITY entityHandle,  // IN: handle of entity
                               TPM2B_NAME*    bind           // OUT: binding value
);

//*** SessionSetStartTime()
// This function is used to initialize the session timing
void SessionSetStartTime(SESSION* session  // IN: the session to update
);

//*** SessionResetPolicyData()
// This function is used to reset the policy data without changing the nonce
// or the start time of the session.
void SessionResetPolicyData(SESSION* session  // IN: the session to reset
);

//*** SessionCapGetLoaded()
// This function returns a list of handles of loaded session, started
// from input 'handle'
//
// 'Handle' must be in valid loaded session handle range, but does not
// have to point to a loaded session.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
SessionCapGetLoaded(TPMI_SH_POLICY handle,     // IN: start handle
                    UINT32         count,      // IN: count of returned handles
                    TPML_HANDLE*   handleList  // OUT: list of handle
);

//*** SessionCapGetOneLoaded()
// This function returns whether a session handle exists and is loaded.
BOOL SessionCapGetOneLoaded(TPMI_SH_POLICY handle  // IN: handle
);

//*** SessionCapGetSaved()
// This function returns a list of handles for saved session, starting at
// 'handle'.
//
// 'Handle' must be in a valid handle range, but does not have to point to a
// saved session
//
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
SessionCapGetSaved(TPMI_SH_HMAC handle,     // IN: start handle
                   UINT32       count,      // IN: count of returned handles
                   TPML_HANDLE* handleList  // OUT: list of handle
);

//*** SessionCapGetOneSaved()
// This function returns whether a session handle exists and is saved.
BOOL SessionCapGetOneSaved(TPMI_SH_HMAC handle  // IN: handle
);

//*** SessionCapGetLoadedNumber()
// This function return the number of authorization sessions currently
// loaded into TPM RAM.
UINT32
SessionCapGetLoadedNumber(void);
```

```
//*** SessionCapGetLoadedAvail()
// This function returns the number of additional authorization sessions, of
// any type, that could be loaded into TPM RAM.
// NOTE: In other implementations, this number may just be an estimate. The only
//       requirement for the estimate is, if it is one or more, then at least one
//       session must be loadable.
UINT32
SessionCapGetLoadedAvail(void);

//*** SessionCapGetActiveNumber()
// This function returns the number of active authorization sessions currently
// being tracked by the TPM.
UINT32
SessionCapGetActiveNumber(void);

//*** SessionCapGetActiveAvail()
// This function returns the number of additional authorization sessions, of any
// type, that could be created. This not the number of slots for sessions, but
// the number of additional sessions that the TPM is capable of tracking.
UINT32
SessionCapGetActiveAvail(void);

#endif  // _SESSION_FP_H_
```

6.205 /tpm/include/private/prototypes/SetAlgorithmSet_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_SetAlgorithmSet  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETALGORITHMSET_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETALGORITHMSET_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PLATFORM authHandle;
    UINT32           algorithmSet;
} SetAlgorithmSet_In;

// Response code modifiers
#    define RC_SetAlgorithmSet_authHandle   (TPM_RC_H + TPM_RC_1)
#    define RC_SetAlgorithmSet_algorithmSet (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_SetAlgorithmSet(SetAlgorithmSet_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETALGORITHMSET_FP_H_
#endif    // CC_SetAlgorithmSet
```

6.206 /tpm/include/private/prototypes/SetCapability_fp.h

```
#if CC_SetCapability  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETCAPABILITY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETCAPABILITY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_HIERARCHY         authHandle;
    TPM2B_SET_CAPABILITY_DATA setCapabilityData;
```

```
} SetCapability_In;

// Response code modifiers
#    define SetCapability_authHandle        (TPM_RC_H + TPM_RC_1)
#    define SetCapability_setCapabilityData (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC TPM2_SetCapability(SetCapability_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETCAPABILITY_FP_H_
#endif    // CC_SetCapability
```

6.207 /tpm/include/private/prototypes/SetCommandCodeAuditStatus_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_SetCommandCodeAuditStatus  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETCOMMANDCODEAUDITSTATUS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETCOMMANDCODEAUDITSTATUS_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_PROVISION auth;
    TPMI_ALG_HASH     auditAlg;
    TPML_CC           setList;
    TPML_CC           clearList;
} SetCommandCodeAuditStatus_In;

// Response code modifiers
#    define RC_SetCommandCodeAuditStatus_auth      (TPM_RC_H + TPM_RC_1)
#    define RC_SetCommandCodeAuditStatus_auditAlg  (TPM_RC_P + TPM_RC_1)
#    define RC_SetCommandCodeAuditStatus_setList   (TPM_RC_P + TPM_RC_2)
#    define RC_SetCommandCodeAuditStatus_clearList (TPM_RC_P + TPM_RC_3)

// Function prototype
TPM_RC
TPM2_SetCommandCodeAuditStatus(SetCommandCodeAuditStatus_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETCOMMANDCODEAUDITSTATUS_FP_H_
#endif    // CC_SetCommandCodeAuditStatus
```

6.208 /tpm/include/private/prototypes/SetPrimaryPolicy_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_SetPrimaryPolicy  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETPRIMARYPOLICY_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETPRIMARYPOLICY_FP_H_

// Input structure definition
typedef struct
{
    TPMI_RH_HIERARCHY_POLICY authHandle;
    TPM2B_DIGEST             authPolicy;
    TPMI_ALG_HASH            hashAlg;
} SetPrimaryPolicy_In;

// Response code modifiers
#    define RC_SetPrimaryPolicy_authHandle (TPM_RC_H + TPM_RC_1)
#    define RC_SetPrimaryPolicy_authPolicy (TPM_RC_P + TPM_RC_1)
#    define RC_SetPrimaryPolicy_hashAlg    (TPM_RC_P + TPM_RC_2)
```

```
// Function prototype
TPM_RC
TPM2_SetPrimaryPolicy(SetPrimaryPolicy_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SETPRIMARYPOLICY_FP_H_
#endif    // CC_SetPrimaryPolicy
```

6.209 /tpm/include/private/prototypes/Shutdown_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Shutdown  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SHUTDOWN_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SHUTDOWN_FP_H_

// Input structure definition
typedef struct
{
    TPM_SU shutdownType;
} Shutdown_In;

// Response code modifiers
#    define RC_Shutdown_shutdownType (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_Shutdown(Shutdown_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SHUTDOWN_FP_H_
#endif    // CC_Shutdown
```

6.210 /tpm/include/private/prototypes/Sign_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Sign  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_SIGN_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_SIGN_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT    keyHandle;
    TPM2B_DIGEST      digest;
    TPMT_SIG_SCHEME   inScheme;
    TPMT_TK_HASHCHECK validation;
} Sign_In;

// Output structure definition
typedef struct
{
    TPMT_SIGNATURE signature;
} Sign_Out;

// Response code modifiers
#    define RC_Sign_keyHandle  (TPM_RC_H + TPM_RC_1)
#    define RC_Sign_digest     (TPM_RC_P + TPM_RC_1)
#    define RC_Sign_inScheme   (TPM_RC_P + TPM_RC_2)
#    define RC_Sign_validation (TPM_RC_P + TPM_RC_3)

// Function prototype
```

```
TPM_RC
TPM2_Sign(Sign_In* in, Sign_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_SIGN_FP_H_
#endif    // CC_Sign
```

6.211 /tpm/include/private/prototypes/StartAuthSession_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_StartAuthSession  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_STARTAUTHSESSION_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_STARTAUTHSESSION_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT          tpmKey;
    TPMI_DH_ENTITY          bind;
    TPM2B_NONCE             nonceCaller;
    TPM2B_ENCRYPTED_SECRET encryptedSalt;
    TPM_SE                  sessionType;
    TPMT_SYM_DEF            symmetric;
    TPMI_ALG_HASH           authHash;
} StartAuthSession_In;

// Output structure definition
typedef struct
{
    TPMI_SH_AUTH_SESSION sessionHandle;
    TPM2B_NONCE          nonceTPM;
} StartAuthSession_Out;

// Response code modifiers
#    define RC_StartAuthSession_tpmKey         (TPM_RC_H + TPM_RC_1)
#    define RC_StartAuthSession_bind           (TPM_RC_H + TPM_RC_2)
#    define RC_StartAuthSession_nonceCaller    (TPM_RC_P + TPM_RC_1)
#    define RC_StartAuthSession_encryptedSalt (TPM_RC_P + TPM_RC_2)
#    define RC_StartAuthSession_sessionType    (TPM_RC_P + TPM_RC_3)
#    define RC_StartAuthSession_symmetric      (TPM_RC_P + TPM_RC_4)
#    define RC_StartAuthSession_authHash       (TPM_RC_P + TPM_RC_5)

// Function prototype
TPM_RC
TPM2_StartAuthSession(StartAuthSession_In* in, StartAuthSession_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_STARTAUTHSESSION_FP_H_
#endif    // CC_StartAuthSession
```

6.212 /tpm/include/private/prototypes/Startup_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Startup  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_STARTUP_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_STARTUP_FP_H_

// Input structure definition
typedef struct
{
    TPM_SU startupType;
} Startup_In;
```

```
// Response code modifiers
#    define RC_Startup_startupType (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_Startup(Startup_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_STARTUP_FP_H_
#endif    // CC_Startup
```

6.213 /tpm/include/private/prototypes/StirRandom_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_StirRandom  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_STIRRANDOM_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_STIRRANDOM_FP_H_

// Input structure definition
typedef struct
{
    TPM2B_SENSITIVE_DATA inData;
} StirRandom_In;

// Response code modifiers
#    define RC_StirRandom_inData (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_StirRandom(StirRandom_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_STIRRANDOM_FP_H_
#endif    // CC_StirRandom
```

6.214 /tpm/include/private/prototypes/TableDrivenMarshal_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:44PM
 */

#ifndef _TABLE_DRIVEN_MARSHAL_FP_H_
#define _TABLE_DRIVEN_MARSHAL_FP_H_

#if TABLE_DRIVEN_MARSHAL

//***UnmarshalUnion()
TPM_RC
UnmarshalUnion(UINT16  typeIndex,  // IN: the thing to unmarshal
               void*   target,     // IN: were the data goes to
               UINT8** buffer,     // IN/OUT: the data source buffer
               INT32*  size,       // IN/OUT: the remaining size
               UINT32  selector);

//*** MarshalUnion()
UINT16
MarshalUnion(UINT16  typeIndex,  // IN: the thing to marshal
             void*   source,     // IN: were the data comes from
             UINT8** buffer,     // IN/OUT: the data source buffer
             INT32*  size,       // IN/OUT: the remaining size
             UINT32  selector    // IN: the union selector
);
```

```
TPM_RC
UnmarshalInteger(int     iSize,   // IN: Number of bytes in the integer
                 void*   target,  // OUT: receives the integer
                 UINT8** buffer,  // IN/OUT: source of the data
                 INT32*  size,    // IN/OUT: amount of data available
                 UINT32* value    // OUT: optional copy of 'target'
);

//*** Unmarshal()
// This is the function that performs unmarshaling of different numbered types. Each
// TPM type has a number. The number is used to lookup the address of the data
// structure that describes how to unmarshal that data type.
//
TPM_RC
Unmarshal(UINT16  typeIndex,  // IN: the thing to marshal
          void*   target,     // IN: were the data goes from
          UINT8** buffer,     // IN/OUT: the data source buffer
          INT32*  size        // IN/OUT: the remaining size
);

//*** Marshal()
// This is the function that drives marshaling of output. Because there is no
// validation of the output, there is a lot less code.
UINT16 Marshal(UINT16  typeIndex,  // IN: the thing to marshal
               void*   source,     // IN: were the data comes from
               UINT8** buffer,     // IN/OUT: the data source buffer
               INT32*  size        // IN/OUT: the remaining size
);
#endif  // TABLE_DRIVEN_MARSHAL

#endif  // _TABLE_DRIVEN_MARSHAL_FP_H_
```

6.215 /tpm/include/private/prototypes/TestParms_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_TestParms  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_TESTPARMS_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_TESTPARMS_FP_H_

// Input structure definition
typedef struct
{
    TPMT_PUBLIC_PARMS parameters;
} TestParms_In;

// Response code modifiers
#    define RC_TestParms_parameters (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_TestParms(TestParms_In* in);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_TESTPARMS_FP_H_
#endif    // CC_TestParms
```

6.216 /tpm/include/private/prototypes/Ticket_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
 */
```

```
#ifndef _TICKET_FP_H_
#define _TICKET_FP_H_

//*** TicketIsSafe()
// This function indicates if producing a ticket is safe.
// It checks if the leading bytes of an input buffer is TPM_GENERATED_VALUE
// or its substring of canonical form.  If so, it is not safe to produce ticket
// for an input buffer claiming to be TPM generated buffer
//  Return Type: BOOL
//      TRUE(1)         safe to produce ticket
//      FALSE(0)        not safe to produce ticket
BOOL TicketIsSafe(TPM2B* buffer);

//*** TicketComputeVerified()
// This function creates a TPMT_TK_VERIFIED ticket.
TPM_RC TicketComputeVerified(
    TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy constant for ticket
    TPM2B_DIGEST*     digest,     // IN: digest
    TPM2B_NAME*       keyName,    // IN: name of key that signed the values
    TPMT_TK_VERIFIED* ticket      // OUT: verified ticket
);

//*** TicketComputeAuth()
// This function creates a TPMT_TK_AUTH ticket.
TPM_RC TicketComputeAuth(
    TPM_ST            type,             // IN: the type of ticket.
    TPMI_RH_HIERARCHY hierarchy,        // IN: hierarchy constant for ticket
    UINT64            timeout,          // IN: timeout
    BOOL              expiresOnReset,   // IN: flag to indicate if ticket expires on
                                        //      TPM Reset
    TPM2B_DIGEST* cpHashA,              // IN: input cpHashA
    TPM2B_NONCE*  policyRef,            // IN: input policyRef
    TPM2B_NAME*   entityName,           // IN: name of entity
    TPMT_TK_AUTH* ticket                // OUT: Created ticket
);

//*** TicketComputeHashCheck()
// This function creates a TPMT_TK_HASHCHECK ticket.
TPM_RC TicketComputeHashCheck(
    TPMI_RH_HIERARCHY  hierarchy,  // IN: hierarchy constant for ticket
    TPM_ALG_ID         hashAlg,    // IN: the hash algorithm for 'digest'
    TPM2B_DIGEST*      digest,     // IN: input digest
    TPMT_TK_HASHCHECK* ticket      // OUT: Created ticket
);

//*** TicketComputeCreation()
// This function creates a TPMT_TK_CREATION ticket.
TPM_RC TicketComputeCreation(TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy for ticket
                             TPM2B_NAME*       name,       // IN: object name
                             TPM2B_DIGEST*     creation,   // IN: creation hash
                             TPMT_TK_CREATION* ticket      // OUT: created ticket
);

#endif  // _TICKET_FP_H_
```

6.217 /tpm/include/private/prototypes/Time_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 04:23:27PM
 */

#ifndef _TIME_FP_H_
#define _TIME_FP_H_
```

```
//*** TimePowerOn()
// This function initialize time info at _TPM_Init().
//
// This function is called at _TPM_Init() so that the TPM time can start counting
// as soon as the TPM comes out of reset and doesn't have to wait until
// TPM2_Startup() in order to begin the new time epoch. This could be significant
// for systems that could get powered up but not run any TPM commands for some
// period of time.
//
void TimePowerOn(void);

//*** TimeStartup()
// This function updates the resetCount and restartCount components of
// TPMS_CLOCK_INFO structure at TPM2_Startup().
//
// This function will deal with the deferred creation of a new epoch.
// TimeUpdateToCurrent() will not start a new epoch even if one is due when
// TPM_Startup() has not been run. This is because the state of NV is not known
// until startup completes. When Startup is done, then it will create the epoch
// nonce to complete the initializations by calling this function.
BOOL TimeStartup(STARTUP_TYPE type  // IN: start up type
);

//*** TimeClockUpdate()
// This function updates go.clock. If 'newTime' requires an update of NV, then
// NV is checked for availability. If it is not available or is rate limiting, then
// go.clock is not updated and the function returns an error. If 'newTime' would
// not cause an NV write, then go.clock is updated. If an NV write occurs, then
// go.safe is SET.
void TimeClockUpdate(UINT64 newTime  // IN: New time value in mS.
);

//*** TimeUpdate()
// This function is used to update the time and clock values. If the TPM
// has run TPM2_Startup(), this function is called at the start of each command.
// If the TPM has not run TPM2_Startup(), this is called from TPM2_Startup() to
// get the clock values initialized. It is not called on command entry because, in
// this implementation, the go structure is not read from NV until TPM2_Startup().
// The reason for this is that the initialization code (_TPM_Init()) may run before
// NV is accessible.
void TimeUpdate(void);

//*** TimeUpdateToCurrent()
// This function updates the 'Time' and 'Clock' in the global
// TPMS_TIME_INFO structure.
//
// In this implementation, 'Time' and 'Clock' are updated at the beginning
// of each command and the values are unchanged for the duration of the
// command.
//
// Because 'Clock' updates may require a write to NV memory, 'Time' and 'Clock'
// are not allowed to advance if NV is not available. When clock is not advancing,
// any function that uses 'Clock' will fail and return TPM_RC_NV_UNAVAILABLE or
// TPM_RC_NV_RATE.
//
// This implementation does not do rate limiting. If the implementation does do
// rate limiting, then the 'Clock' update should not be inhibited even when doing
// rate limiting.
void TimeUpdateToCurrent(void);

//*** TimeSetAdjustRate()
// This function is used to perform rate adjustment on 'Time' and 'Clock'.
void TimeSetAdjustRate(TPM_CLOCK_ADJUST adjust  // IN: adjust constant
);
```

```
//*** TimeGetMarshaled()
// This function is used to access TPMS_TIME_INFO in canonical form.
// The function collects the time information and marshals it into 'dataBuffer'
// and returns the marshaled size
UINT16
TimeGetMarshaled(TIME_INFO* dataBuffer  // OUT: result buffer
);

//*** TimeFillInfo
// This function gathers information to fill in a TPMS_CLOCK_INFO structure.
void TimeFillInfo(TPMS_CLOCK_INFO* clockInfo);

#endif  // _TIME_FP_H_
```

6.218 /tpm/include/private/prototypes/TpmASN1_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Aug 30, 2019  Time: 02:11:54PM
 */

#ifndef _TPM_ASN1_FP_H_
#define _TPM_ASN1_FP_H_

//*** ASN1UnmarshalContextInitialize()
// Function does standard initialization of a context.
//  Return Type: BOOL
//      TRUE(1)     success
//      FALSE(0)    failure
BOOL ASN1UnmarshalContextInitialize(
    ASN1UnmarshalContext* ctx, INT16 size, BYTE* buffer);

//***ASN1DecodeLength()
// This function extracts the length of an element from 'buffer' starting at 'offset'.
// Return Type: UINT16
//      >=0         the extracted length
//      <0          an error
INT16
ASN1DecodeLength(ASN1UnmarshalContext* ctx);

//***ASN1NextTag()
// This function extracts the next type from 'buffer' starting at 'offset'.
// It advances 'offset' as it parses the type and the length of the type. It returns
// the length of the type. On return, the 'length' octets starting at 'offset' are the
// octets of the type.
// Return Type: UINT
//     >=0          the number of octets in 'type'
//     <0           an error
INT16
ASN1NextTag(ASN1UnmarshalContext* ctx);

//*** ASN1GetBitStringValue()
// Try to parse a bit string of up to 32 bits from a value that is expected to be
// a bit string. The bit string is left justified so that the MSb of the input is
// the MSb of the returned value.
// If there is a general parsing error, the context->size is set to -1.
//  Return Type: BOOL
//      TRUE(1)     success
//      FALSE(0)    failure
BOOL ASN1GetBitStringValue(ASN1UnmarshalContext* ctx, UINT32* val);

//*** ASN1InitializeMarshalContext()
// This creates a structure for handling marshaling of an ASN.1 formatted data
// structure.
void ASN1InitializeMarshalContext(
```

```
    ASN1MarshalContext* ctx, INT16 length, BYTE* buffer);

//*** ASN1StartMarshalContext()
// This starts a new constructed element. It is constructed on 'top' of the value
// that was previously placed in the structure.
void ASN1StartMarshalContext(ASN1MarshalContext* ctx);

//*** ASN1EndMarshalContext()
// This function restores the end pointer for an encapsulating structure.
//  Return Type: INT16
//      > 0             the size of the encapsulated structure that was just ended
//      <= 0            an error
INT16
ASN1EndMarshalContext(ASN1MarshalContext* ctx);

//***ASN1EndEncapsulation()
// This function puts a tag and length in the buffer. In this function, an embedded
// BIT_STRING is assumed to be a collection of octets. To indicate that all bits
// are used, a byte of zero is prepended. If a raw bit-string is needed, a new
// function like ASN1PushInteger() would be needed.
//  Return Type: INT16
//      > 0         number of octets in the encapsulation
//      == 0        failure
UINT16
ASN1EndEncapsulation(ASN1MarshalContext* ctx, BYTE tag);

//*** ASN1PushByte()
BOOL ASN1PushByte(ASN1MarshalContext* ctx, BYTE b);

//*** ASN1PushBytes()
// Push some raw bytes onto the buffer. 'count' cannot be zero.
//  Return Type: IN16
//      > 0             count bytes
//      == 0            failure unless count was zero
INT16
ASN1PushBytes(ASN1MarshalContext* ctx, INT16 count, const BYTE* buffer);

//*** ASN1PushNull()
//  Return Type: IN16
//      > 0             count bytes
//      == 0            failure unless count was zero
INT16
ASN1PushNull(ASN1MarshalContext* ctx);

//*** ASN1PushLength()
// Push a length value. This will only handle length values that fit in an INT16.
//  Return Type: UINT16
//      > 0         number of bytes added
//      == 0        failure
INT16
ASN1PushLength(ASN1MarshalContext* ctx, INT16 len);

//*** ASN1PushTagAndLength()
//  Return Type: INT16
//      > 0         number of bytes added
//      == 0        failure
INT16
ASN1PushTagAndLength(ASN1MarshalContext* ctx, BYTE tag, INT16 length);

//*** ASN1PushTaggedOctetString()
// This function will push a random octet string.
//  Return Type: INT16
//      > 0         number of bytes added
//      == 0        failure
INT16
ASN1PushTaggedOctetString(
```

```
    ASN1MarshalContext* ctx, INT16 size, const BYTE* string, BYTE tag);

//*** ASN1PushUINT()
// This function pushes an native-endian integer value. This just changes a
// native-endian integer into a big-endian byte string and calls ASN1PushInteger().
// That function will remove leading zeros and make sure that the number is positive.
//  Return Type: IN16
//      > 0             count bytes
//      == 0            failure unless count was zero
INT16
ASN1PushUINT(ASN1MarshalContext* ctx, UINT32 integer);

//*** ASN1PushInteger
// Push a big-endian integer on the end of the buffer
//  Return Type: UINT16
//      > 0         the number of bytes marshaled for the integer
//      == 0        failure
INT16
ASN1PushInteger(ASN1MarshalContext* ctx,     // IN/OUT: buffer context
                INT16               iLen,    // IN: octets of the integer
                BYTE*               integer  // IN: big-endian integer
);

//*** ASN1PushOID()
// This function is used to add an OID. An OID is 0x06 followed by a byte of size
// followed by size bytes. This is used to avoid having to do anything special in the
// definition of an OID.
//  Return Type: UINT16
//      > 0         the number of bytes marshaled for the integer
//      == 0        failure
INT16
ASN1PushOID(ASN1MarshalContext* ctx, const BYTE* OID);

#endif  // _TPM_ASN1_FP_H_
```

6.219 /tpm/include/private/prototypes/TpmEcc_Signature_ECDAA_fp.h

```
#ifndef _TPMECC_SIGNATURE_ECDAA_FP_H_
#define _TPMECC_SIGNATURE_ECDAA_FP_H_
#if ALG_ECC && ALG_ECDAA

//*** TpmEcc_SignEcdaa()
//
// This function performs 's' = 'r' + 'T' * 'd' mod 'q' where
// 1) 'r' is a random, or pseudo-random value created in the commit phase
// 2) 'nonceK' is a TPM-generated, random value 0 < 'nonceK' < 'n'
// 3) 'T' is mod 'q' of "Hash"('nonceK' || 'digest'), and
// 4) 'd' is a private key.
//
// The signature is the tuple ('nonceK', 's')
//
// Regrettably, the parameters in this function kind of collide with the parameter
// names used in ECSCHNORR making for a lot of confusion.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       unsupported hash algorithm
//      TPM_RC_NO_RESULT    cannot get values from random number generator
TPM_RC TpmEcc_SignEcdaa(
    TPM2B_ECC_PARAMETER*  nonceK,  // OUT: 'nonce' component of the signature
    Crypt_Int*            bnS,     // OUT: 's' component of the signature
    const Crypt_EccCurve* E,       // IN: the curve used in signing
    Crypt_Int*            bnD,     // IN: the private key
    const TPM2B_DIGEST*   digest,  // IN: the value to sign (mod 'q')
    TPMT_ECC_SCHEME*      scheme,  // IN: signing scheme (contains the
                                   //      commit count value).
    OBJECT*     eccKey,            // IN: The signing key
```

```
    RAND_STATE* rand                // IN: a random number state
);

#endif  // ALG_ECC && ALG_ECDAA
#endif  // _TPMECC_SIGNATURE_ECDAA_FP_H_
```

6.220 /tpm/include/private/prototypes/TpmEcc_Signature_ECDSA_fp.h

```
#ifndef _TPMECC_SIGNATURE_ECDSA_FP_H_
#define _TPMECC_SIGNATURE_ECDSA_FP_H_
#if ALG_ECC && ALG_ECDSA

//*** TpmEcc_SignEcdsa()
// This function implements the ECDSA signing algorithm. The method is described
// in the comments below.
TPM_RC
TpmEcc_SignEcdsa(Crypt_Int*            bnR,    // OUT: 'r' component of the signature
                 Crypt_Int*            bnS,    // OUT: 's' component of the signature
                 const Crypt_EccCurve* E,      // IN: the curve used in the signature
                                               //     process
                 Crypt_Int*          bnD,      // IN: private signing key
                 const TPM2B_DIGEST* digest,   // IN: the digest to sign
                 RAND_STATE*         rand      // IN: used in debug of signing
);

//*** TpmEcc_ValidateSignatureEcdsa()
// This function validates an ECDSA signature. rIn and sIn should have been checked
// to make sure that they are in the range 0 < 'v' < 'n'
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE           signature not valid
TPM_RC
TpmEcc_ValidateSignatureEcdsa(
    Crypt_Int*            bnR,  // IN: 'r' component of the signature
    Crypt_Int*            bnS,  // IN: 's' component of the signature
    const Crypt_EccCurve* E,    // IN: the curve used in the signature
                                //     process
    const Crypt_Point*  ecQ,    // IN: the public point of the key
    const TPM2B_DIGEST* digest  // IN: the digest that was signed
);

#endif  // ALG_ECC && ALG_ECDSA
#endif  // _TPMECC_SIGNATURE_ECDSA_FP_H_
```

6.221 /tpm/include/private/prototypes/TpmEcc_Signature_Schnorr_fp.h

```
#ifndef _TPMECC_SIGNATURE_SCHNORR_FP_H_
#define _TPMECC_SIGNATURE_SCHNORR_FP_H_

#if ALG_ECC && ALG_ECSCHNORR
TPM_RC TpmEcc_SignEcSchnorr(
    Crypt_Int*            bnR,      // OUT: 'r' component of the signature
    Crypt_Int*            bnS,      // OUT: 's' component of the signature
    const Crypt_EccCurve* E,        // IN: the curve used in signing
    Crypt_Int*            bnD,      // IN: the signing key
    const TPM2B_DIGEST*   digest,   // IN: the digest to sign
    TPM_ALG_ID            hashAlg,  // IN: signing scheme (contains a hash)
    RAND_STATE*           rand      // IN: non-NULL when testing
);

//*** TpmEcc_ValidateSignatureEcSchnorr()
// This function is used to validate an EC Schnorr signature.
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE        signature not valid
TPM_RC TpmEcc_ValidateSignatureEcSchnorr(
```

```
    Crypt_Int*            bnR,      // IN: 'r' component of the signature
    Crypt_Int*            bnS,      // IN: 's' component of the signature
    TPM_ALG_ID            hashAlg,  // IN: hash algorithm of the signature
    const Crypt_EccCurve* E,        // IN: the curve used in the signature
                                    //     process
    Crypt_Point*        ecQ,        // IN: the public point of the key
    const TPM2B_DIGEST* digest      // IN: the digest that was signed
);

#endif  // ALG_ECC && ALG_ECSCHNORR
#endif  // _TPMECC_SIGNATURE_SCHNORR_FP_H_
```

6.222 /tpm/include/private/prototypes/TpmEcc_Signature_SM2_fp.h

```
#ifndef _TPMECC_SIGNATURE_SM2_FP_H_
#define _TPMECC_SIGNATURE_SM2_FP_H_

#if ALG_ECC && ALG_SM2
//*** TpmEcc_SignEcSm2()
// This function signs a digest using the method defined in SM2 Part 2. The method
// in the standard will add a header to the message to be signed that is a hash of
// the values that define the key. This then hashed with the message to produce a
// digest ('e'). This function signs 'e'.
//  Return Type: TPM_RC
//      TPM_RC_VALUE         bad curve
TPM_RC TpmEcc_SignEcSm2(Crypt_Int* bnR,  // OUT: 'r' component of the signature
                        Crypt_Int* bnS,  // OUT: 's' component of the signature
                        const Crypt_EccCurve* E,    // IN: the curve used in signing
                        Crypt_Int*            bnD,  // IN: the private key
                        const TPM2B_DIGEST*   digest,  // IN: the digest to sign
                        RAND_STATE* rand  // IN: random number generator (mostly for
                                          //     debug)
);

//*** TpmEcc_ValidateSignatureEcSm2()
// This function is used to validate an SM2 signature.
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE            signature not valid
TPM_RC TpmEcc_ValidateSignatureEcSm2(
    Crypt_Int*            bnR,  // IN: 'r' component of the signature
    Crypt_Int*            bnS,  // IN: 's' component of the signature
    const Crypt_EccCurve* E,    // IN: the curve used in the signature
                                //     process
    Crypt_Point*        ecQ,    // IN: the public point of the key
    const TPM2B_DIGEST* digest  // IN: the digest that was signed
);

#endif  // ALG_ECC && ALG_SM2
#endif  // _TPMECC_SIGNATURE_SM2_FP_H_
```

6.223 /tpm/include/private/prototypes/TpmEcc_Signature_Util_fp.h

```
// functions shared by multiple signature algorithms
#ifndef _TPMECC_SIGNATURE_UTIL_FP_H_
#define _TPMECC_SIGNATURE_UTIL_FP_H_

#if ALG_ECC
//*** TpmEcc_SchnorrCalculateS()
// This contains the Schnorr signature (S) computation. It is used by both ECDSA and
// Schnorr signing. The result is computed as: ['s' = 'k' + 'r' * 'd' (mod 'n')]
// where
// 1) 's' is the signature
// 2) 'k' is a random value
// 3) 'r' is the value to sign
```

```
// 4) 'd' is the private EC key
// 5) 'n' is the order of the curve
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        the result of the operation was zero or 'r' (mod 'n')
//                              is zero
TPM_RC TpmEcc_SchnorrCalculateS(
    Crypt_Int*       bnS,  // OUT: 's' component of the signature
    const Crypt_Int* bnK,  // IN: a random value
    Crypt_Int*       bnR,  // IN: the signature 'r' value
    const Crypt_Int* bnD,  // IN: the private key
    const Crypt_Int* bnN   // IN: the order of the curve
);

#endif  // ALG_ECC
#endif  // _TPMECC_SIGNATURE_UTIL_FP_H_
```

6.224 /tpm/include/private/prototypes/TpmEcc_Util_fp.h

```
#ifndef _TPMECC_UTIL_FP_H_
#define _TPMECC_UTIL_FP_H_

#if ALG_ECC

//*** TpmEcc_PointFrom2B()
// Function to create a Crypt_Point structure from a 2B point.
// This function doesn't take an Crypt_EccCurve for legacy reasons -
// this should probably be changed.
// returns NULL if the input value is invalid or doesn't fit.
LIB_EXPORT Crypt_Point* TpmEcc_PointFrom2B(
    Crypt_Point*    ecP,  // OUT: the preallocated point structure
    TPMS_ECC_POINT* p     // IN: the number to convert
);

//*** TpmEcc_PointTo2B()
// This function converts a Crypt_Point into a TPMS_ECC_POINT. A TPMS_ECC_POINT
// contains two TPM2B_ECC_PARAMETER values. The maximum size of the parameters
// is dependent on the maximum EC key size used in an implementation.
// The presumption is that the TPMS_ECC_POINT is large enough to hold 2 TPM2B
// values, each as large as a MAX_ECC_PARAMETER_BYTES
LIB_EXPORT BOOL TpmEcc_PointTo2B(
    TPMS_ECC_POINT*       p,    // OUT: the converted 2B structure
    const Crypt_Point*    ecP,  // IN: the values to be converted
    const Crypt_EccCurve* E     // IN: curve descriptor for the point
);

#endif  // ALG_ECC
#endif  // _TPMECC_UTIL_FP_H_
```

6.225 /tpm/include/private/prototypes/TpmMath_Debug_fp.h

```
//
// debug and test utilities.  Not expected to be compiled into final products
#ifndef _TPMMATH_DEBUG_FP_H_
#define _TPMMATH_DEBUG_FP_H_

#if ALG_ECC || ALG_RSA

//*** TpmEccDebug_HexEqual()
// This function compares a bignum value to a hex string.
// using TpmEcc namespace because code assumes the max size
// is correct for ECC.
//  Return Type: BOOL
//      TRUE(1)         values equal
//      FALSE(0)        values not equal
```

```
BOOL TpmMath_Debug_HexEqual(const Crypt_Int* bn,  //IN: big number value
                            const char*      c    //IN: character string number
);

LIB_EXPORT Crypt_Int* TpmMath_Debug_FromHex(
    Crypt_Int*           bn,         // OUT:
    const unsigned char* hex,        // IN:
    size_t               maxsizeHex  // IN: maximum size of hex
);

#endif  // ALG_ECC or ALG_RSA
#endif  //_TPMMATH_DEBUG_FP_H_
```

6.226 /tpm/include/private/prototypes/TpmMath_Util_fp.h

```
#ifndef _TPM_MATH_FP_H_
#define _TPM_MATH_FP_H_

//*** TpmMath_IntFrom2B()
// Convert an TPM2B to a Crypt_Int.
// If the input value does not exist, or the output does not exist, or the input
// will not fit into the output the function returns NULL
LIB_EXPORT Crypt_Int* TpmMath_IntFrom2B(Crypt_Int*   value,  // OUT:
                                        const TPM2B* a2B     // IN: number to convert
);

//*** TpmMath_IntTo2B()
//
// Function to convert a Crypt_Int to TPM2B. The TPM2B bytes are
// always in big-endian ordering (most significant byte first). If 'size' is
// non-zero and less than required by `value` then an error is returned. If
// `size` is non-zero and larger than `value`, the result buffer is padded
// with zeros. If `size` is zero, then the TPM2B is assumed to be large enough
// for the data and a2b->size will be adjusted accordingly.
LIB_EXPORT BOOL TpmMath_IntTo2B(
    const Crypt_Int* value,  // IN: value to convert
    TPM2B*           a2B,    // OUT: buffer for output
    NUMBYTES         size    // IN: Size of output buffer - see comments.
);

//*** TpmMath_GetRandomBits()
// This function gets random bits for use in various places.
//
// One consequence of the generation scheme is that, if the number of bits requested
// is not a multiple of 8, then the high-order bits are set to zero. This would come
// into play when generating a 521-bit ECC key. A 66-byte (528-bit) value is
// generated and the high order 7 bits are masked off (CLEAR).
// In this situation, the highest order byte is the first byte (big-endian/TPM2B
format)
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
LIB_EXPORT BOOL TpmMath_GetRandomBits(
    BYTE*       pBuffer,  // OUT: buffer to set
    size_t      bits,     // IN: number of bits to generate (see remarks)
    RAND_STATE* rand      // IN: random engine
);

//*** TpmMath_GetRandomInteger
// This function generates a random integer with the requested number of bits.
// Except for size, no range checking is performed.
// The maximum size that can be created is LARGEST_NUMBER + 64 bits.
// if either more bits, or the Crypt_Int* is too small to contain the requested bits
// the TPM enters failure mode and this function returns FALSE.
LIB_EXPORT BOOL TpmMath_GetRandomInteger(Crypt_Int* bn,  // OUT: integer buffer to set
```

```
                                        size_t     bits,  // IN: size of output,
                                        RAND_STATE* rand  // IN: random engine
);

//*** TpmMath_GetRandomInRange()
// This function is used to generate a random number r in the range 1 <= r < limit.
// The function gets a random number of bits that is the size of limit. There is some
// some probability that the returned number is going to be greater than or equal
// to the limit. If it is, try again. There is no more than 50% chance that the
// next number is also greater, so try again. We keep trying until we get a
// value that meets the criteria. Since limit is very often a number with a LOT of
// high order ones, this rarely would need a second try.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure ('limit' is too small)
LIB_EXPORT BOOL TpmMath_GetRandomInRange(
    Crypt_Int*       dest,   // OUT: integer buffer to set
    const Crypt_Int* limit,  // IN: limit (see remarks)
    RAND_STATE*      rand    // IN: random engine
);

#endif  //_TPM_MATH_FP_H_
```

6.227 /tpm/include/private/prototypes/TpmSizeChecks_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Oct 24, 2019  Time: 11:37:07AM
 */

#ifndef _TPM_SIZE_CHECKS_FP_H_
#define _TPM_SIZE_CHECKS_FP_H_

#if RUNTIME_SIZE_CHECKS

//** TpmSizeChecks()
// This function is used during the development process to make sure that the
// vendor-specific values result in a consistent implementation. When possible,
// the code contains "#if" to do compile-time checks. However, in some cases, the
// values require the use of "sizeof()" and that can't be used in an #if.
BOOL TpmSizeChecks(void);
#endif  // RUNTIME_SIZE_CHECKS

#endif  // _TPM_SIZE_CHECKS_FP_H_
```

6.228 /tpm/include/private/prototypes/Unseal_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Unseal  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_UNSEAL_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_UNSEAL_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT itemHandle;
} Unseal_In;

// Output structure definition
typedef struct
{
    TPM2B_SENSITIVE_DATA outData;
```

```
} Unseal_Out;

// Response code modifiers
#    define RC_Unseal_itemHandle (TPM_RC_H + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_Unseal(Unseal_In* in, Unseal_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_UNSEAL_FP_H_
#endif    // CC_Unseal
```

6.229 /tpm/include/private/prototypes/Vendor_TCG_Test_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_Vendor_TCG_Test  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_VENDOR_TCG_TEST_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_VENDOR_TCG_TEST_FP_H_

// Input structure definition
typedef struct
{
    TPM2B_DATA inputData;
} Vendor_TCG_Test_In;

// Output structure definition
typedef struct
{
    TPM2B_DATA outputData;
} Vendor_TCG_Test_Out;

// Response code modifiers
#    define RC_Vendor_TCG_Test_inputData (TPM_RC_P + TPM_RC_1)

// Function prototype
TPM_RC
TPM2_Vendor_TCG_Test(Vendor_TCG_Test_In* in, Vendor_TCG_Test_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_VENDOR_TCG_TEST_FP_H_
#endif    // CC_Vendor_TCG_Test
```

6.230 /tpm/include/private/prototypes/VerifySignature_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_VerifySignature  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_VERIFYSIGNATURE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_VERIFYSIGNATURE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT keyHandle;
    TPM2B_DIGEST   digest;
    TPMT_SIGNATURE signature;
} VerifySignature_In;

// Output structure definition
typedef struct
{
    TPMT_TK_VERIFIED validation;
```

```
} VerifySignature_Out;

// Response code modifiers
#    define RC_VerifySignature_keyHandle (TPM_RC_H + TPM_RC_1)
#    define RC_VerifySignature_digest    (TPM_RC_P + TPM_RC_1)
#    define RC_VerifySignature_signature (TPM_RC_P + TPM_RC_2)

// Function prototype
TPM_RC
TPM2_VerifySignature(VerifySignature_In* in, VerifySignature_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_VERIFYSIGNATURE_FP_H_
#endif    // CC_VerifySignature
```

6.231 /tpm/include/private/prototypes/X509_ECC_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 11:00:49AM
 */

#ifndef _X509_ECC_FP_H_
#define _X509_ECC_FP_H_

//*** X509PushPoint()
// This seems like it might be used more than once so...
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509PushPoint(ASN1MarshalContext* ctx, TPMS_ECC_POINT* p);

//*** X509AddSigningAlgorithmECC()
// This creates the singing algorithm data.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddSigningAlgorithmECC(
    OBJECT* signKey, TPMT_SIG_SCHEME* scheme, ASN1MarshalContext* ctx);

//*** X509AddPublicECC()
// This function will add the publicKey description to the DER data. If ctx is
// NULL, then no data is transferred and this function will indicate if the TPM
// has the values for DER-encoding of the public key.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddPublicECC(OBJECT* object, ASN1MarshalContext* ctx);

#endif  // _X509_ECC_FP_H_
```

6.232 /tpm/include/private/prototypes/X509_RSA_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 11:00:49AM
 */

#ifndef _X509_RSA_FP_H_
#define _X509_RSA_FP_H_

#if ALG_RSA
```

```
//*** X509AddSigningAlgorithmRSA()
// This creates the singing algorithm data.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddSigningAlgorithmRSA(
    OBJECT* signKey, TPMT_SIG_SCHEME* scheme, ASN1MarshalContext* ctx);

//*** X509AddPublicRSA()
// This function will add the publicKey description to the DER data. If fillPtr is
// NULL, then no data is transferred and this function will indicate if the TPM
// has the values for DER-encoding of the public key.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddPublicRSA(OBJECT* object, ASN1MarshalContext* ctx);
#endif  // ALG_RSA

#endif  // _X509_RSA_FP_H_
```

6.233 /tpm/include/private/prototypes/X509_spt_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Nov 14, 2019  Time: 05:57:02PM
 */

#ifndef _X509_SPT_FP_H_
#define _X509_SPT_FP_H_

//*** X509FindExtensionByOID()
// This will search a list of X509 extensions to find an extension with the
// requested OID. If the extension is found, the output context ('ctx') is set up
// to point to the OID in the extension.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure (could be catastrophic)
BOOL X509FindExtensionByOID(ASN1UnmarshalContext* ctxIn,  // IN: the context to search
                            ASN1UnmarshalContext* ctx,  // OUT: the extension context
                            const BYTE*           OID   // IN: oid to search for
);

//*** X509GetExtensionBits()
// This function will extract a bit field from an extension. If the extension doesn't
// contain a bit string, it will fail.
// Return Type: BOOL
//  TRUE(1)         success
//  FALSE(0)        failure
UINT32
X509GetExtensionBits(ASN1UnmarshalContext* ctx, UINT32* value);

//***X509ProcessExtensions()
// This function is used to process the TPMA_OBJECT and KeyUsage extensions. It is not
// in the CertifyX509.c code because it makes the code harder to follow.
// Return Type: TPM_RC
//      TPM_RCS_ATTRIBUTES      the attributes of object are not consistent with
//                              the extension setting
//      TPM_RC_VALUE            problem parsing the extensions
TPM_RC
X509ProcessExtensions(
    OBJECT* object,         // IN: The object with the attributes to
                            //      check
```

```
    stringRef* extension  // IN: The start and length of the extensions
);

//*** X509AddSigningAlgorithm()
// This creates the singing algorithm data.
// Return Type: INT16
//  > 0                 number of octets added
// <= 0                 failure
INT16
X509AddSigningAlgorithm(
    ASN1MarshalContext* ctx, OBJECT* signKey, TPMT_SIG_SCHEME* scheme);

//*** X509AddPublicKey()
// This function will add the publicKey description to the DER data. If fillPtr is
// NULL, then no data is transferred and this function will indicate if the TPM
// has the values for DER-encoding of the public key.
//  Return Type: INT16
//      > 0         number of octets added
//      == 0        failure
INT16
X509AddPublicKey(ASN1MarshalContext* ctx, OBJECT* object);

//*** X509PushAlgorithmIdentifierSequence()
// The function adds the algorithm identifier sequence.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509PushAlgorithmIdentifierSequence(ASN1MarshalContext* ctx, const BYTE* OID);

#endif  // _X509_SPT_FP_H_
```

6.234 /tpm/include/private/prototypes/ZGen_2Phase_fp.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#if CC_ZGen_2Phase  // Command must be enabled

#  ifndef _TPM_INCLUDE_PRIVATE_PROTOTYPES_ZGEN_2PHASE_FP_H_
#    define _TPM_INCLUDE_PRIVATE_PROTOTYPES_ZGEN_2PHASE_FP_H_

// Input structure definition
typedef struct
{
    TPMI_DH_OBJECT        keyA;
    TPM2B_ECC_POINT       inQsB;
    TPM2B_ECC_POINT       inQeB;
    TPMI_ECC_KEY_EXCHANGE inScheme;
    UINT16                counter;
} ZGen_2Phase_In;

// Output structure definition
typedef struct
{
    TPM2B_ECC_POINT outZ1;
    TPM2B_ECC_POINT outZ2;
} ZGen_2Phase_Out;

// Response code modifiers
#    define RC_ZGen_2Phase_keyA     (TPM_RC_H + TPM_RC_1)
#    define RC_ZGen_2Phase_inQsB    (TPM_RC_P + TPM_RC_1)
#    define RC_ZGen_2Phase_inQeB    (TPM_RC_P + TPM_RC_2)
#    define RC_ZGen_2Phase_inScheme (TPM_RC_P + TPM_RC_3)
#    define RC_ZGen_2Phase_counter  (TPM_RC_P + TPM_RC_4)
```

```
// Function prototype
TPM_RC
TPM2_ZGen_2Phase(ZGen_2Phase_In* in, ZGen_2Phase_Out* out);

#  endif  // _TPM_INCLUDE_PRIVATE_PROTOTYPES_ZGEN_2PHASE_FP_H_
#endif    // CC_ZGen_2Phase
```

6.235 /tpm/include/public/ACT.h

```
#ifndef _ACT_H_
#define _ACT_H_

#include <TpmConfiguration/TpmProfile.h>

#if ACT_SUPPORT                                                                 \
    != (RH_ACT_0 | RH_ACT_1 | RH_ACT_2 | RH_ACT_3 | RH_ACT_4 | RH_ACT_5 | RH_ACT_6   \
        | RH_ACT_7 | RH_ACT_8 | RH_ACT_9 | RH_ACT_A | RH_ACT_B | RH_ACT_C | RH_ACT_D \
        | RH_ACT_E | RH_ACT_F)
#  error "If ACT_SUPPORT == NO, no ACTs can be enabled"
#endif  // (ACT_SUPPORT != ...)

#if !(defined RH_ACT_0) || (RH_ACT_0 != YES)
#  undef RH_ACT_0
#  define RH_ACT_0 NO
#  define IF_ACT_0_IMPLEMENTED(op)
#else
#  define IF_ACT_0_IMPLEMENTED(op) op(0)
#endif
#if !(defined RH_ACT_1) || (RH_ACT_1 != YES)
#  undef RH_ACT_1
#  define RH_ACT_1 NO
#  define IF_ACT_1_IMPLEMENTED(op)
#else
#  define IF_ACT_1_IMPLEMENTED(op) op(1)
#endif
#if !(defined RH_ACT_2) || (RH_ACT_2 != YES)
#  undef RH_ACT_2
#  define RH_ACT_2 NO
#  define IF_ACT_2_IMPLEMENTED(op)
#else
#  define IF_ACT_2_IMPLEMENTED(op) op(2)
#endif
#if !(defined RH_ACT_3) || (RH_ACT_3 != YES)
#  undef RH_ACT_3
#  define RH_ACT_3 NO
#  define IF_ACT_3_IMPLEMENTED(op)
#else
#  define IF_ACT_3_IMPLEMENTED(op) op(3)
#endif
#if !(defined RH_ACT_4) || (RH_ACT_4 != YES)
#  undef RH_ACT_4
#  define RH_ACT_4 NO
#  define IF_ACT_4_IMPLEMENTED(op)
#else
#  define IF_ACT_4_IMPLEMENTED(op) op(4)
#endif
#if !(defined RH_ACT_5) || (RH_ACT_5 != YES)
#  undef RH_ACT_5
#  define RH_ACT_5 NO
#  define IF_ACT_5_IMPLEMENTED(op)
#else
#  define IF_ACT_5_IMPLEMENTED(op) op(5)
#endif
#if !(defined RH_ACT_6) || (RH_ACT_6 != YES)
#  undef RH_ACT_6
```

```
#  define RH_ACT_6 NO
#  define IF_ACT_6_IMPLEMENTED(op)
#else
#  define IF_ACT_6_IMPLEMENTED(op) op(6)
#endif
#if !(defined RH_ACT_7) || (RH_ACT_7 != YES)
#  undef RH_ACT_7
#  define RH_ACT_7 NO
#  define IF_ACT_7_IMPLEMENTED(op)
#else
#  define IF_ACT_7_IMPLEMENTED(op) op(7)
#endif
#if !(defined RH_ACT_8) || (RH_ACT_8 != YES)
#  undef RH_ACT_8
#  define RH_ACT_8 NO
#  define IF_ACT_8_IMPLEMENTED(op)
#else
#  define IF_ACT_8_IMPLEMENTED(op) op(8)
#endif
#if !(defined RH_ACT_9) || (RH_ACT_9 != YES)
#  undef RH_ACT_9
#  define RH_ACT_9 NO
#  define IF_ACT_9_IMPLEMENTED(op)
#else
#  define IF_ACT_9_IMPLEMENTED(op) op(9)
#endif
#if !(defined RH_ACT_A) || (RH_ACT_A != YES)
#  undef RH_ACT_A
#  define RH_ACT_A NO
#  define IF_ACT_A_IMPLEMENTED(op)
#else
#  define IF_ACT_A_IMPLEMENTED(op) op(A)
#endif
#if !(defined RH_ACT_B) || (RH_ACT_B != YES)
#  undef RH_ACT_B
#  define RH_ACT_B NO
#  define IF_ACT_B_IMPLEMENTED(op)
#else
#  define IF_ACT_B_IMPLEMENTED(op) op(B)
#endif
#if !(defined RH_ACT_C) || (RH_ACT_C != YES)
#  undef RH_ACT_C
#  define RH_ACT_C NO
#  define IF_ACT_C_IMPLEMENTED(op)
#else
#  define IF_ACT_C_IMPLEMENTED(op) op(C)
#endif
#if !(defined RH_ACT_D) || (RH_ACT_D != YES)
#  undef RH_ACT_D
#  define RH_ACT_D NO
#  define IF_ACT_D_IMPLEMENTED(op)
#else
#  define IF_ACT_D_IMPLEMENTED(op) op(D)
#endif
#if !(defined RH_ACT_E) || (RH_ACT_E != YES)
#  undef RH_ACT_E
#  define RH_ACT_E NO
#  define IF_ACT_E_IMPLEMENTED(op)
#else
#  define IF_ACT_E_IMPLEMENTED(op) op(E)
#endif
#if !(defined RH_ACT_F) || (RH_ACT_F != YES)
#  undef RH_ACT_F
#  define RH_ACT_F NO
#  define IF_ACT_F_IMPLEMENTED(op)
#else
```

```
#  define IF_ACT_F_IMPLEMENTED(op) op(F)
#endif

#ifndef TPM_RH_ACT_0
#  error Need numeric definition for TPM_RH_ACT_0
#endif

#ifndef TPM_RH_ACT_1
#  define TPM_RH_ACT_1 (TPM_RH_ACT_0 + 1)
#endif
#ifndef TPM_RH_ACT_2
#  define TPM_RH_ACT_2 (TPM_RH_ACT_0 + 2)
#endif
#ifndef TPM_RH_ACT_3
#  define TPM_RH_ACT_3 (TPM_RH_ACT_0 + 3)
#endif
#ifndef TPM_RH_ACT_4
#  define TPM_RH_ACT_4 (TPM_RH_ACT_0 + 4)
#endif
#ifndef TPM_RH_ACT_5
#  define TPM_RH_ACT_5 (TPM_RH_ACT_0 + 5)
#endif
#ifndef TPM_RH_ACT_6
#  define TPM_RH_ACT_6 (TPM_RH_ACT_0 + 6)
#endif
#ifndef TPM_RH_ACT_7
#  define TPM_RH_ACT_7 (TPM_RH_ACT_0 + 7)
#endif
#ifndef TPM_RH_ACT_8
#  define TPM_RH_ACT_8 (TPM_RH_ACT_0 + 8)
#endif
#ifndef TPM_RH_ACT_9
#  define TPM_RH_ACT_9 (TPM_RH_ACT_0 + 9)
#endif
#ifndef TPM_RH_ACT_A
#  define TPM_RH_ACT_A (TPM_RH_ACT_0 + 0xA)
#endif
#ifndef TPM_RH_ACT_B
#  define TPM_RH_ACT_B (TPM_RH_ACT_0 + 0xB)
#endif
#ifndef TPM_RH_ACT_C
#  define TPM_RH_ACT_C (TPM_RH_ACT_0 + 0xC)
#endif
#ifndef TPM_RH_ACT_D
#  define TPM_RH_ACT_D (TPM_RH_ACT_0 + 0xD)
#endif
#ifndef TPM_RH_ACT_E
#  define TPM_RH_ACT_E (TPM_RH_ACT_0 + 0xE)
#endif
#ifndef TPM_RH_ACT_F
#  define TPM_RH_ACT_F (TPM_RH_ACT_0 + 0xF)
#endif

#define FOR_EACH_ACT(op)      \
    IF_ACT_0_IMPLEMENTED(op) \
    IF_ACT_1_IMPLEMENTED(op) \
    IF_ACT_2_IMPLEMENTED(op) \
    IF_ACT_3_IMPLEMENTED(op) \
    IF_ACT_4_IMPLEMENTED(op) \
    IF_ACT_5_IMPLEMENTED(op) \
    IF_ACT_6_IMPLEMENTED(op) \
    IF_ACT_7_IMPLEMENTED(op) \
    IF_ACT_8_IMPLEMENTED(op) \
    IF_ACT_9_IMPLEMENTED(op) \
    IF_ACT_A_IMPLEMENTED(op) \
    IF_ACT_B_IMPLEMENTED(op) \
```

```
    IF_ACT_C_IMPLEMENTED(op) \
    IF_ACT_D_IMPLEMENTED(op) \
    IF_ACT_E_IMPLEMENTED(op) \
    IF_ACT_F_IMPLEMENTED(op)

// This is the mask for ACT that are implemented
//#define ACT_MASK(N)      | (1 << 0x##N)
//#define ACT_IMPLEMENTED_MASK    (0 FOR_EACH_ACT(ACT_MASK))

#define CASE_ACT_HANDLE(N) case TPM_RH_ACT_##N:
#define CASE_ACT_NUMBER(N) case 0x##N:

typedef struct ACT_STATE
{
    UINT32       remaining;
    TPM_ALG_ID   hashAlg;
    TPM2B_DIGEST authPolicy;
} ACT_STATE, *P_ACT_STATE;

#endif  // _ACT_H_
```

6.236 /tpm/include/public/BaseTypes.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#ifndef _TPM_INCLUDE_PUBLIC_BASETYPES_H_
#define _TPM_INCLUDE_PUBLIC_BASETYPES_H_

// NULL definition
#ifndef NULL
#  define NULL (0)
#endif  // NULL

typedef uint8_t  UINT8;
typedef uint8_t  BYTE;
typedef int8_t   INT8;
typedef int      BOOL;
typedef uint16_t UINT16;
typedef int16_t  INT16;
typedef uint32_t UINT32;
typedef int32_t  INT32;
typedef uint64_t UINT64;
typedef int64_t  INT64;

#endif  // _TPM_INCLUDE_PUBLIC_BASETYPES_H_
```

6.237 /tpm/include/public/Capabilities.h

```
#ifndef _CAPABILITIES_H
#define _CAPABILITIES_H

#define MAX_CAP_DATA         (MAX_CAP_BUFFER - sizeof(TPM_CAP) - sizeof(UINT32))
#define MAX_CAP_ALGS         (MAX_CAP_DATA / sizeof(TPMS_ALG_PROPERTY))
#define MAX_CAP_HANDLES      (MAX_CAP_DATA / sizeof(TPM_HANDLE))
#define MAX_CAP_CC           (MAX_CAP_DATA / sizeof(TPM_CC))
#define MAX_TPM_PROPERTIES   (MAX_CAP_DATA / sizeof(TPMS_TAGGED_PROPERTY))
#define MAX_PCR_PROPERTIES   (MAX_CAP_DATA / sizeof(TPMS_TAGGED_PCR_SELECT))
#define MAX_ECC_CURVES       (MAX_CAP_DATA / sizeof(TPM_ECC_CURVE))
#define MAX_TAGGED_POLICIES  (MAX_CAP_DATA / sizeof(TPMS_TAGGED_POLICY))
#define MAX_ACT_DATA         (MAX_CAP_DATA / sizeof(TPMS_ACT_DATA))
#define MAX_AC_CAPABILITIES  (MAX_CAP_DATA / sizeof(TPMS_AC_OUTPUT))

#endif
```

6.238 /tpm/include/public/CompilerDependencies.h

```
// This file contains the build switches. This contains switches for multiple
// versions of the crypto-library so some may not apply to your environment.
//

#ifndef _COMPILER_DEPENDENCIES_H_
#define _COMPILER_DEPENDENCIES_H_

#if defined(__GNUC__)
#  include <public/CompilerDependencies_gcc.h>
#elif defined(_MSC_VER)
#  include <public/CompilerDependencies_msvc.h>
#else
#  error unexpected
#endif

#include <stdint.h>

// Things that are not defined should be defined as NULL

#ifndef NORETURN
#  define NORETURN
#endif
#ifndef LIB_EXPORT
#  define LIB_EXPORT
#endif
#ifndef LIB_IMPORT
#  define LIB_IMPORT
#endif
#ifndef _REDUCE_WARNING_LEVEL_
#  define _REDUCE_WARNING_LEVEL_(n)
#endif
#ifndef _NORMAL_WARNING_LEVEL_
#  define _NORMAL_WARNING_LEVEL_
#endif
#ifndef NOT_REFERENCED
#  define NOT_REFERENCED(x) (x = x)
#endif

#ifdef _POSIX_
typedef int SOCKET;
#endif

#if !defined(TPM_STATIC_ASSERT) || !defined(COMPILER_CHECKS)
#  error Expect definitions of COMPILER_CHECKS and TPM_STATIC_ASSERT
#elif COMPILER_CHECKS
// pre static_assert static_assert
#  define MUST_BE(e) TPM_STATIC_ASSERT(e)

#else
// intentionally disabled, fine.
#  define MUST_BE(e)
#endif

#endif  // _COMPILER_DEPENDENCIES_H_
```

6.239 /tpm/include/public/CompilerDependencies_gcc.h

```
// This file contains compiler specific switches.
// These definitions are for the GCC compiler
//

#ifndef _COMPILER_DEPENDENCIES_GCC_H_
#define _COMPILER_DEPENDENCIES_GCC_H_
```

```
#if !defined(__GNUC__)
#  error CompilerDependencies_gcc.h included for wrong compiler
#endif

// don't warn on unused local typedefs, they are used as a
// cross-compiler static_assert
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-local-typedefs"
#pragma GCC diagnostic pop

#undef _MSC_VER
#undef WIN32

#ifndef WINAPI
#  define WINAPI
#endif
#ifndef __pragma
#  define __pragma(x)
#endif
#define REVERSE_ENDIAN_16(_Number) __builtin_bswap16(_Number)
#define REVERSE_ENDIAN_32(_Number) __builtin_bswap32(_Number)
#define REVERSE_ENDIAN_64(_Number) __builtin_bswap64(_Number)

#define NORETURN __attribute__((noreturn))

#define TPM_INLINE           inline __attribute__((always_inline))
#define TPM_STATIC_ASSERT(e) _Static_assert(e, "static assert")
#endif  // _COMPILER_DEPENDENCIES_H_
```

6.240 /tpm/include/public/CompilerDependencies_msvc.h

```
// This file contains compiler specific switches.
// These definitions are for the Microsoft compiler
//

#ifndef _COMPILER_DEPENDENCIES_MSVC_H_
#define _COMPILER_DEPENDENCIES_MSVC_H_

#if !defined(_MSC_VER)
#  error CompilerDependencies_msvc.h included for wrong compiler
#endif

// Endian conversion for aligned structures
#define REVERSE_ENDIAN_16(_Number) _byteswap_ushort(_Number)
#define REVERSE_ENDIAN_32(_Number) _byteswap_ulong(_Number)
#define REVERSE_ENDIAN_64(_Number) _byteswap_uint64(_Number)

// Avoid compiler warning for in line of stdio (or not)
//#define _NO_CRT_STDIO_INLINE

// This macro is used to handle LIB_EXPORT of function and variable names in lieu
// of a .def file. Visual Studio requires that functions be explicitly exported and
// imported.
#ifdef TPM_AS_DLL
#  define LIB_EXPORT __declspec(dllexport)  // VS compatible version
#  define LIB_IMPORT __declspec(dllimport)
#else
// building static libraries
#  define LIB_EXPORT
#  define LIB_IMPORT
#endif

#define TPM_INLINE inline
```

```
// This is defined to indicate a function that does not return. Microsoft compilers
// do not support the _Noretrun function parameter.
#define NORETURN __declspec(noreturn)
#if _MSC_VER >= 1400  // SAL processing when needed
#  include <sal.h>
#endif

// #  ifdef _WIN64
// #    define _INTPTR 2
// #  else
// #    define _INTPTR 1
// #  endif

#define NOT_REFERENCED(x) (x)

// Lower the compiler error warning for system include
// files. They tend not to be that clean and there is no
// reason to sort through all the spurious errors that they
// generate when the normal error level is set to /Wall
#define _REDUCE_WARNING_LEVEL_(n) __pragma(warning(push, n))
// Restore the compiler warning level
#define _NORMAL_WARNING_LEVEL_ __pragma(warning(pop))
#include <stdint.h>

#ifdef TPM_STATIC_ASSERT
#  error TPM_STATIC_ASSERT already defined
#endif

// MSVC: failure results in error C2118: negative subscript error
#define TPM_STATIC_ASSERT(e) typedef char __C_ASSERT__[(e) ? 1 : -1]

#endif  // _COMPILER_DEPENDENCIES_MSVC_H_
```

6.241 /tpm/include/public/endian_swap.h

```
#ifndef _SWAP_H
#define _SWAP_H

#if LITTLE_ENDIAN_TPM
#  define TO_BIG_ENDIAN_UINT16(i)   REVERSE_ENDIAN_16(i)
#  define FROM_BIG_ENDIAN_UINT16(i) REVERSE_ENDIAN_16(i)
#  define TO_BIG_ENDIAN_UINT32(i)   REVERSE_ENDIAN_32(i)
#  define FROM_BIG_ENDIAN_UINT32(i) REVERSE_ENDIAN_32(i)
#  define TO_BIG_ENDIAN_UINT64(i)   REVERSE_ENDIAN_64(i)
#  define FROM_BIG_ENDIAN_UINT64(i) REVERSE_ENDIAN_64(i)
#else
#  define TO_BIG_ENDIAN_UINT16(i)   (i)
#  define FROM_BIG_ENDIAN_UINT16(i) (i)
#  define TO_BIG_ENDIAN_UINT32(i)   (i)
#  define FROM_BIG_ENDIAN_UINT32(i) (i)
#  define TO_BIG_ENDIAN_UINT64(i)   (i)
#  define FROM_BIG_ENDIAN_UINT64(i) (i)
#endif

#if AUTO_ALIGN == NO

// The aggregation macros for machines that do not allow unaligned access or for
// little-endian machines.

// Aggregate bytes into an UINT

#  define BYTE_ARRAY_TO_UINT8(b)      (uint8_t)((b)[0])
#  define BYTE_ARRAY_TO_UINT16(b)     ByteArrayToUint16((BYTE*)(b))
#  define BYTE_ARRAY_TO_UINT32(b)     ByteArrayToUint32((BYTE*)(b))
#  define BYTE_ARRAY_TO_UINT64(b)     ByteArrayToUint64((BYTE*)(b))
```

```
#  define UINT8_TO_BYTE_ARRAY(i, b)   ((b)[0] = (uint8_t)(i))
#  define UINT16_TO_BYTE_ARRAY(i, b) Uint16ToByteArray((i), (BYTE*)(b))
#  define UINT32_TO_BYTE_ARRAY(i, b) Uint32ToByteArray((i), (BYTE*)(b))
#  define UINT64_TO_BYTE_ARRAY(i, b) Uint64ToByteArray((i), (BYTE*)(b))

#else  // AUTO_ALIGN

#  if BIG_ENDIAN_TPM
// the big-endian macros for machines that allow unaligned memory access
// Aggregate a byte array into a UINT
#    define BYTE_ARRAY_TO_UINT8(b)  *((uint8_t*)(b))
#    define BYTE_ARRAY_TO_UINT16(b) *((uint16_t*)(b))
#    define BYTE_ARRAY_TO_UINT32(b) *((uint32_t*)(b))
#    define BYTE_ARRAY_TO_UINT64(b) *((uint64_t*)(b))

// Disaggregate a UINT into a byte array

#    define UINT8_TO_BYTE_ARRAY(i, b) \
        {                             \
            *((uint8_t*)(b)) = (i);   \
        }
#    define UINT16_TO_BYTE_ARRAY(i, b) \
        {                              \
            *((uint16_t*)(b)) = (i);   \
        }
#    define UINT32_TO_BYTE_ARRAY(i, b) \
        {                              \
            *((uint32_t*)(b)) = (i);   \
        }
#    define UINT64_TO_BYTE_ARRAY(i, b) \
        {                              \
            *((uint64_t*)(b)) = (i);   \
        }
#  else
// the little endian macros for machines that allow unaligned memory access
// the big-endian macros for machines that allow unaligned memory access
// Aggregate a byte array into a UINT
#    define BYTE_ARRAY_TO_UINT8(b)  *((uint8_t*)(b))
#    define BYTE_ARRAY_TO_UINT16(b) REVERSE_ENDIAN_16(*((uint16_t*)(b)))
#    define BYTE_ARRAY_TO_UINT32(b) REVERSE_ENDIAN_32(*((uint32_t*)(b)))
#    define BYTE_ARRAY_TO_UINT64(b) REVERSE_ENDIAN_64(*((uint64_t*)(b)))

// Disaggregate a UINT into a byte array

#    define UINT8_TO_BYTE_ARRAY(i, b) \
        {                             \
            *((uint8_t*)(b)) = (i);   \
        }
#    define UINT16_TO_BYTE_ARRAY(i, b)                \
        {                                             \
            *((uint16_t*)(b)) = REVERSE_ENDIAN_16(i); \
        }
#    define UINT32_TO_BYTE_ARRAY(i, b)                \
        {                                             \
            *((uint32_t*)(b)) = REVERSE_ENDIAN_32(i); \
        }
#    define UINT64_TO_BYTE_ARRAY(i, b)                \
        {                                             \
            *((uint64_t*)(b)) = REVERSE_ENDIAN_64(i); \
        }
#  endif  // BIG_ENDIAN_TPM

#endif  // AUTO_ALIGN == NO

#endif  // _SWAP_H
```

6.242 /tpm/include/public/GpMacros.h

```
//** Introduction
// This file is a collection of miscellaneous macros.

#ifndef GP_MACROS_H
#define GP_MACROS_H

#ifndef NULL
#  define NULL 0
#endif

#include "endian_swap.h"
#include <TpmConfiguration/VendorInfo.h>

//** For Self-test
// These macros are used in CryptUtil to invoke the incremental self test.
#if ENABLE_SELF_TESTS
#  define TPM_DO_SELF_TEST(alg)              \
      do                                     \
      {                                      \
          if(TEST_BIT(alg, g_toTest))        \
              CryptTestAlgorithm(alg, NULL); \
      } while(0)
#else
#  define TPM_DO_SELF_TEST(alg)
#endif  // ENABLE_SELF_TESTS

//** For Failures
#if defined _POSIX_
#  define FUNCTION_NAME 0
#else
#  define FUNCTION_NAME __FUNCTION__
#endif

#if defined(FAIL_TRACE) && FAIL_TRACE != 0
#  define CODELOCATOR() FUNCTION_NAME, __LINE__
#else  // !FAIL_TRACE
// if provided, use the definition of CODELOCATOR from TpmConfiguration so
// implementor can customize this.
#  ifndef CODELOCATOR
#    define CODELOCATOR() 0
#  endif
#endif  // FAIL_TRACE

// SETFAILED calls TpmFail.  It may or may not return based on the NO_LONGJMP flag.
// CODELOCATOR is a macro that expands to either one 64-bit value that encodes the
// location, or two parameters: Function Name and Line Number.
#define SETFAILED(errorCode) (TpmFail(CODELOCATOR(), errorCode))

// If implementation is using longjmp, then calls to TpmFail() will never
// return.  However, without longjmp facility, TpmFail will return while most of
// the code currently expects FAIL() calls to immediately abort the current
// command. If they don't, some commands return success instead of failure.  The
// family of macros below are provided to allow the code to be modified to
// correctly propagate errors correctly, based on the context.
//
// * Some functions, particularly the ECC crypto have state cleanup at the end
//   of the function and need to use the goto Exit pattern.
// * Other functions return TPM_RC values, which should return TPM_RC_FAILURE
// * Still other functions return an isOK boolean and need to return FALSE.
//
// if longjmp is available, all these macros just call SETFAILED and immediately
// abort.  Note any of these approaches could leak memory if the crypto adapter
// libraries are using dynamic memory.
//
```

```
// FAIL vs. FAIL_NORET
// ===================
// Be cautious with these macros.  FAIL_NORET is intended as an affirmation
// that the upstream code calling the function using this macro has been
// investigated to confirm that upstream functions correctly handle this
// function putting the TPM into failure mode without returning an error.
//
// The TPM library was originally written with a lot of error checking omitted,
// which means code occurring after a FAIL macro may not expect to be called
// when the TPM is in failure mode.  When NO_LONGJMP is false (the system has a
// longjmp API), then none of that code is executed because the sample platform
// sets up longjmp before calling ExecuteCommand.  However, in the NO_LONGJMP
// case, code following a FAIL or FAIL_NORET macro will get run.  The
// conservative assumption is that code is untested and may be unsafe in such a
// situation.  FAIL_NORET can replace FAIL when the code has been reviewed to
// ensure the post-FAIL code is safe.  Of course, this is a point-in-time
// assertion that is only true when the FAIL_NORET macro is first inserted;
// hence it is better to use one of the early-exit macros to immediately return.
// However, the necessary return-code plumbing may be large and FAIL/FAIL_NORET
// are provided to support gradual improvement over time.

#ifndef NO_LONGJMP
// has longjmp
// necesary to reference Exit, even though the code is no-return
#  define TPM_FAIL_RETURN NORETURN void

// see discussion above about FAIL/FAIL_NORET
#  define FAIL(failCode)                    SETFAILED(failCode)
#  define FAIL_NORET(failCode)              SETFAILED(failCode)
#  define FAIL_IMMEDIATE(failCode, retval) SETFAILED(failCode)
#  define FAIL_BOOL(failCode)               SETFAILED(failCode)
#  define FAIL_RC(failCode)                 SETFAILED(failCode)
#  define FAIL_VOID(failCode)               SETFAILED(failCode)
#  define FAIL_NULL(failCode)               SETFAILED(failCode)
#  define FAIL_EXIT(failCode, returnVar, returnCode)  \
      do                                              \
      {                                               \
          SETFAILED(failCode);                        \
          goto Exit;                                  \
      } while(0)

#else  // NO_LONGJMP
// no longjmp service is available
#  define TPM_FAIL_RETURN      void

// This macro is provided for existing code and should not be used in new code.
// see discussion above.
#  define FAIL(failCode)       FAIL_NORET(failCode)

// Be cautious with this macro, see discussion above.
#  define FAIL_NORET(failCode) SETFAILED(failCode)

// fail and immediately return void
#  define FAIL_VOID(failCode)  \
      do                       \
      {                        \
          SETFAILED(failCode); \
          return;              \
      } while(0)

// fail and immediately return a value
#  define FAIL_IMMEDIATE(failCode, retval) \
      do                                   \
      {                                    \
          SETFAILED(failCode);             \
          return retval;                   \
```

```
    } while(0)

// fail and return FALSE
#  define FAIL_BOOL(failCode) FAIL_IMMEDIATE(failCode, FALSE)

// fail and return TPM_RC_FAILURE
#  define FAIL_RC(failCode)   FAIL_IMMEDIATE(failCode, TPM_RC_FAILURE)

// fail and return NULL
#  define FAIL_NULL(failCode) FAIL_IMMEDIATE(failCode, NULL)

// fail and return using the goto exit pattern
#  define FAIL_EXIT(failCode, returnVar, returnCode) \
      do                                             \
      {                                              \
          SETFAILED(failCode);                       \
          returnVar = returnCode;                    \
          goto Exit;                                 \
      } while(0)

#endif

// This macro tests that a condition is TRUE and puts the TPM into failure mode
// if it is not. If longjmp is being used, then the macro makes a call from
// which there is no return. Otherwise, the function will return the given
// return code.
#define VERIFY(condition, failCode, returnCode)    \
    do                                             \
    {                                              \
        if(!(condition))                           \
        {                                          \
            FAIL_IMMEDIATE(failCode, returnCode); \
        }                                          \
    } while(0)

// this function also verifies a condition and enters failure mode, but sets a
// return value and jumps to Exit on failure - allowing for cleanup.
#define VERIFY_OR_EXIT(condition, failCode, returnVar, returnCode) \
    do                                                             \
    {                                                              \
        if(!(condition))                                           \
        {                                                          \
            FAIL_EXIT(failCode, returnVar, returnCode);            \
        }                                                          \
    } while(0)

// verify the given TPM_RC is success and we are not in
// failure mode.  Otherwise, return immediately with TPM_RC_FAILURE.
// note that failure mode is checked first so that an existing FATAL_* error code
// is not overwritten with the default from this macro.
#define VERIFY_RC(rc)                                          \
    do                                                         \
    {                                                          \
        if(g_inFailureMode)                                    \
        {                                                      \
            return TPM_RC_FAILURE;                             \
        }                                                      \
        if(rc != TPM_RC_SUCCESS)                               \
        {                                                      \
            FAIL_IMMEDIATE(FATAL_ERROR_ASSERT, TPM_RC_FAILURE); \
        }                                                      \
    } while(0)

// verify the TPM is not in failure mode or return failure
#define VERIFY_NOT_FAILED()        \
    do                             \
```

```
    {                                   \
        if(g_inFailureMode)             \
        {                               \
            return TPM_RC_FAILURE;      \
        }                               \
    } while(0)

// Enter failure mode if the given TPM_RC is not success, return void.
#define VERIFY_RC_VOID(rc)                  \
    do                                      \
    {                                       \
        if(g_inFailureMode)                 \
        {                                   \
            return;                         \
        }                                   \
        if(rc != TPM_RC_SUCCESS)            \
        {                                   \
            FAIL_VOID(FATAL_ERROR_ASSERT);  \
        }                                   \
    } while(0)

// These VERIFY_CRYPTO macros all set failure mode to FATAL_ERROR_CRYPTO
// and immediately return.  The general way to parse the names is:
// VERIFY_CRYPTO_[conditionType]_[OR_EXIT]_[retValType]
// if conditionType is omitted, it is taken as BOOL.
// Without OR_EXIT, implies an immediate return. Thus VERIFY_CRYPTO_BOOL:
// 1. check fn against TRUE
// 2. if false,  set failure mode to FATAL_ERROR_CRYPTO
// 3. immediately return FALSE.
// and, VERIFY_CRYPTO_OR_EXIT_RC translates to:
// 1. Check a BOOL
// 2. If false, set failure mode with FATAL_ERROR_CRYPTO,
// 3. assume retVal is type TPM_RC, set it to TPM_RC_FAILURE
// 4. Goto Exit
// while VERIFY_CRYPTO_RC_OR_EXIT translates to:
// 1. Check fn result against TPM_RC_SUCCESS
// 2. if not equal, set failure mode to FATAL_ERROR_CRYPTO
// 3. assume retVal is type TPM_RC, set it to TPM_RC_FAILURE
// 4. Goto Exit.
#define VERIFY_CRYPTO(fn) VERIFY((fn), FATAL_ERROR_CRYPTO, TPM_RC_FAILURE)

#define VERIFY_CRYPTO_BOOL(fn) VERIFY((fn), FATAL_ERROR_CRYPTO, FALSE)

#define VERIFY_CRYPTO_OR_NULL(fn) VERIFY((fn), FATAL_ERROR_CRYPTO, NULL)

// these VERIFY_CRYPTO macros all set a result value and goto Exit
#define VERIFY_CRYPTO_OR_EXIT(fn, returnVar, returnCode) \
    VERIFY_OR_EXIT(fn, FATAL_ERROR_CRYPTO, returnVar, returnCode);

// these VERIFY_CRYPTO_OR_EXIT functions assume the return value variable is
// named retVal
#define VERIFY_CRYPTO_OR_EXIT_RC(fn) \
    VERIFY_CRYPTO_OR_EXIT_GENERIC(fn, retVal, TPM_RC_FAILURE)

#define VERIFY_CRYPTO_OR_EXIT_FALSE(fn) \
    VERIFY_CRYPTO_OR_EXIT_GENERIC(fn, retVal, FALSE)

#define VERIFY_CRYPTO_RC_OR_EXIT(fn)                    \
    do                                                  \
    {                                                   \
        TPM_RC rc = fn;                                 \
        if(rc != TPM_RC_SUCCESS)                        \
        {                                               \
            FAIL_EXIT(FATAL_ERROR_CRYPTO, retVal, rc);  \
        }                                               \
    } while(0)
```

```
#if(defined EMPTY_ASSERT) && (EMPTY_ASSERT != NO)
#  define pAssert(a) ((void)0)
#else
#  define pAssert(a)                          \
      do                                      \
      {                                       \
          if(!(a))                            \
              FAIL(FATAL_ERROR_PARAMETER); \
      } while(0)

#  define pAssert_ZERO(a)                                 \
      do                                                  \
      {                                                   \
          if(!(a))                                        \
              FAIL_IMMEDIATE(FATAL_ERROR_ASSERT, 0); \
      } while(0);

#  define pAssert_RC(a)                       \
      do                                      \
      {                                       \
          if(!(a))                            \
              FAIL_RC(FATAL_ERROR_ASSERT); \
      } while(0);

#  define pAssert_BOOL(a)                       \
      do                                        \
      {                                         \
          if(!(a))                              \
              FAIL_BOOL(FATAL_ERROR_ASSERT); \
      } while(0);

#  define pAssert_NULL(a)                       \
      do                                        \
      {                                         \
          if(!(a))                              \
              FAIL_NULL(FATAL_ERROR_ASSERT); \
      } while(0);

// using FAIL_NORET isn't optimium but is available in limited cases that
// result in wrong calculated values, and can be checked later
// but should have no vulnerability implications.
#  define pAssert_NORET(a)                       \
      {                                          \
          if(!(a))                               \
              FAIL_NORET(FATAL_ERROR_ASSERT); \
      }

// this macro is used where a calling code has been verified to function correctly
// when the failing assert immediately returns without an error code.
// this can be because either the caller checks the fatal error flag, or
// the state is safe and a higher-level check will catch it.
#  define pAssert_VOID_OK(a)                    \
      {                                         \
          if(!(a))                              \
              FAIL_VOID(FATAL_ERROR_ASSERT); \
      }

#endif

// These macros are commonly used in the "Crypt" code as a way to keep listings from
// getting too long. This is not to save paper but to allow one to see more
// useful stuff on the screen at any given time.  Neither macro sets failure mode.
#define ERROR_EXIT(returnCode)  \
    do                          \
    {                           \
```

```
        retVal = returnCode;    \
        goto Exit;              \
    } while(0)

// braces are necessary for this usage:
// if (y)
//     GOTO_ERROR_UNLESS(x)
// else ...
// without braces the else would attach to the GOTO macro instead of the
// outer if statement; given the amount of TPM code that doesn't use braces on
// if statements, this is a live risk.
#define GOTO_ERROR_UNLESS(_X) \
    do                        \
    {                         \
        if(!(_X))             \
            goto Error;       \
    } while(0)

#include "public/MinMax.h"

#ifndef IsOdd
#  define IsOdd(a) (((a) & 1) != 0)
#endif

#ifndef BITS_TO_BYTES
#  define BITS_TO_BYTES(bits) (((bits) + 7) >> 3)
#endif

// These are defined for use when the size of the vector being checked is known
// at compile time.
#define TEST_BIT(bit, vector)  TestBit((bit), (BYTE*)&(vector), sizeof(vector))
#define SET_BIT(bit, vector)   SetBit((bit), (BYTE*)&(vector), sizeof(vector))
#define CLEAR_BIT(bit, vector) ClearBit((bit), (BYTE*)&(vector), sizeof(vector))

// The following definitions are used if they have not already been defined. The
// defaults for these settings are compatible with ISO/IEC 9899:2011 (E)
#ifndef LIB_EXPORT
#  define LIB_EXPORT
#  define LIB_IMPORT
#endif
#ifndef NORETURN
#  define NORETURN _Noreturn
#endif
#ifndef NOT_REFERENCED
#  define NOT_REFERENCED(x = x) ((void)(x))
#endif

#define STD_RESPONSE_HEADER (sizeof(TPM_ST) + sizeof(UINT32) + sizeof(TPM_RC))

// This bit is used to indicate that an authorization ticket expires on TPM Reset
// and TPM Restart. It is added to the timeout value returned by TPM2_PoliySigned()
// and TPM2_PolicySecret() and used by TPM2_PolicyTicket(). The timeout value is
// relative to Time (g_time). Time is reset whenever the TPM loses power and cannot
// be moved forward by the user (as can Clock). 'g_time' is a 64-bit value expressing
// time in ms. Stealing the MSb for a flag means that the TPM needs to be reset
// at least once every 292,471,208 years rather than once every 584,942,417 years.
#define EXPIRATION_BIT ((UINT64)1 << 63)

// Check for consistency of the bit ordering of bit fields
#if BIG_ENDIAN_TPM && MOST_SIGNIFICANT_BIT_0 && USE_BIT_FIELD_STRUCTURES
#  error "Settings not consistent"
#endif

// These macros are used to handle the variation in handling of bit fields. If
#if USE_BIT_FIELD_STRUCTURES  // The default, old version, with bit fields
#  define IS_ATTRIBUTE(a, type, b)    ((a.b) != 0)
```

```
#  define SET_ATTRIBUTE(a, type, b)    (a.b = SET)
#  define CLEAR_ATTRIBUTE(a, type, b)  (a.b = CLEAR)
#  define GET_ATTRIBUTE(a, type, b)    (a.b)
#  define TPMA_ZERO_INITIALIZER() \
      {                           \
          0                       \
      }
#else
#  define IS_ATTRIBUTE(a, type, b)     ((a & type##_##b) != 0)
#  define SET_ATTRIBUTE(a, type, b)    (a |= type##_##b)
#  define CLEAR_ATTRIBUTE(a, type, b)  (a &= ~type##_##b)
#  define GET_ATTRIBUTE(a, type, b)    (type)((a & type##_##b) >> type##_##b##_SHIFT)
#  define TPMA_ZERO_INITIALIZER()      (0)
#endif

// These macros determine if the values in this file are referenced or instanced.
// Global.c defines GLOBAL_C so all the values in this file will be instanced in
// Global.obj. For all other files that include this file, the values will simply
// be external references. For constants, there can be an initializer.
#ifndef EXTERN
#  ifdef GLOBAL_C
#    define EXTERN
#  else
#    define EXTERN extern
#  endif
#endif  // EXTERN

#ifdef GLOBAL_C
#  define INITIALIZER(_value_) = _value_
#else
#  define INITIALIZER(_value_)
#endif

// This macro will create an OID. All OIDs are in DER form with a first octet of
// 0x06 indicating an OID fallowed by an octet indicating the number of octets in the
// rest of the OID. This allows a user of this OID to know how much/little to copy.
#define MAKE_OID(NAME) EXTERN const BYTE OID##NAME[] INITIALIZER({OID##NAME##_VALUE})

// This definition is moved from TpmProfile.h because it is not actually vendor-
// specific. It has to be the same size as the 'sequence' parameter of a TPMS_CONTEXT
// and that is a UINT64. So, this is an invariant value
#define CONTEXT_COUNTER UINT64

#include "public/TpmCalculatedAttributes.h"

#endif  // GP_MACROS_H
```

6.243 /tpm/include/public/MinMax.h

```
#ifndef _MIN_MAX_H_
#define _MIN_MAX_H_

#ifndef MAX
#  define MAX(a, b) ((a) > (b) ? (a) : (b))
#endif
#ifndef MIN
#  define MIN(a, b) ((a) < (b) ? (a) : (b))
#endif

#ifndef SIZEOF_MEMBER
#  define SIZEOF_MEMBER(type, member) sizeof(((type*)0)->member)
#endif

#endif  // _MIN_MAX_H_
```

6.244 /tpm/include/public/TpmAlgorithmDefines.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#ifndef _TPM_INCLUDE_PRIVATE_TPMALGORITHMDEFINES_H_
#define _TPM_INCLUDE_PRIVATE_TPMALGORITHMDEFINES_H_

#include <TpmConfiguration/TpmProfile.h>
#include "public/MinMax.h"
#include "public/TPMB.h"

#if ALG_ECC
// Table "Defines for NIST_P192 ECC Values" (TCG Algorithm Registry)
#  define NIST_P192_ID       TPM_ECC_NIST_P192
#  define NIST_P192_KEY_SIZE 192

// Table "Defines for NIST_P224 ECC Values" (TCG Algorithm Registry)
#  define NIST_P224_ID       TPM_ECC_NIST_P224
#  define NIST_P224_KEY_SIZE 224

// Table "Defines for NIST_P256 ECC Values" (TCG Algorithm Registry)
#  define NIST_P256_ID       TPM_ECC_NIST_P256
#  define NIST_P256_KEY_SIZE 256

// Table "Defines for NIST_P384 ECC Values" (TCG Algorithm Registry)
#  define NIST_P384_ID       TPM_ECC_NIST_P384
#  define NIST_P384_KEY_SIZE 384

// Table "Defines for NIST_P521 ECC Values" (TCG Algorithm Registry)
#  define NIST_P521_ID       TPM_ECC_NIST_P521
#  define NIST_P521_KEY_SIZE 521

// Table "Defines for BN_P256 ECC Values" (TCG Algorithm Registry)
#  define BN_P256_ID       TPM_ECC_BN_P256
#  define BN_P256_KEY_SIZE 256

// Table "Defines for BN_P638 ECC Values" (TCG Algorithm Registry)
#  define BN_P638_ID       TPM_ECC_BN_P638
#  define BN_P638_KEY_SIZE 638

// Table "Defines for SM2_P256 ECC Values" (TCG Algorithm Registry)
#  define SM2_P256_ID       TPM_ECC_SM2_P256
#  define SM2_P256_KEY_SIZE 256

// Table "Defines for BP_P256_R1 ECC Values" (TCG Algorithm Registry)
#  define BP_P256_R1_ID       TPM_ECC_BP_P256_R1
#  define BP_P256_R1_KEY_SIZE 256

// Table "Defines for BP_P384_R1 ECC Values" (TCG Algorithm Registry)
#  define BP_P384_R1_ID       TPM_ECC_BP_P384_R1
#  define BP_P384_R1_KEY_SIZE 384

// Table "Defines for BP_P512_R1 ECC Values" (TCG Algorithm Registry)
#  define BP_P512_R1_ID       TPM_ECC_BP_P512_R1
#  define BP_P512_R1_KEY_SIZE 512

// Table "Defines for CURVE_25519 ECC Values" (TCG Algorithm Registry)
#  define CURVE_25519_ID       TPM_ECC_CURVE_25519
#  define CURVE_25519_KEY_SIZE 256

// Table "Defines for CURVE_448 ECC Values" (TCG Algorithm Registry)
#  define CURVE_448_ID       TPM_ECC_CURVE_448
#  define CURVE_448_KEY_SIZE 448

// Derived ECC Value
#  define ECC_CURVES                                                    \
```

```
    {                                                                       \
        TPM_ECC_NIST_P192, TPM_ECC_NIST_P224, TPM_ECC_NIST_P256,            \
            TPM_ECC_NIST_P384, TPM_ECC_NIST_P521, TPM_ECC_BN_P256,          \
            TPM_ECC_BN_P638, TPM_ECC_SM2_P256, TPM_ECC_BP_P256_R1,          \
            TPM_ECC_BP_P384_R1, TPM_ECC_BP_P512_R1, TPM_ECC_CURVE_25519,    \
            TPM_ECC_CURVE_448                                               \
    }

#  define ECC_CURVE_COUNT                                                          \
     (ECC_NIST_P192 + ECC_NIST_P224 + ECC_NIST_P256 + ECC_NIST_P384 + ECC_NIST_P521 \
      + ECC_BN_P256 + ECC_BN_P638 + ECC_SM2_P256 + ECC_BP_P256_R1 + ECC_BP_P384_R1  \
      + ECC_BP_P512_R1 + ECC_CURVE_25519 + ECC_CURVE_448)

// Avoid expanding MAX_ECC_KEY_BITS into a long expression, the compiler slows down
// and on some compilers runs out of heap space.

// 638
#  if ECC_BN_P638
#    define MAX_ECC_KEY_BITS BN_P638_KEY_SIZE
// 521
#  elif ECC_NIST_P521
#    define MAX_ECC_KEY_BITS NIST_P521_KEY_SIZE
#  elif ECC_BP_P512_R1
#    define MAX_ECC_KEY_BITS BP_P512_R1_KEY_SIZE
// 448
#  elif ECC_CURVE_448
#    define MAX_ECC_KEY_BITS CURVE_448_KEY_SIZE
// 384
#  elif ECC_NIST_P384
#    define MAX_ECC_KEY_BITS NIST_P384_KEY_SIZE
#  elif ECC_BP_P384_R1
#    define MAX_ECC_KEY_BITS BP_P384_R1_KEY_SIZE
// 256
#  elif ECC_NIST_P256
#    define MAX_ECC_KEY_BITS NIST_P256_KEY_SIZE
#  elif TPM_ECC_BN_P256
#    define MAX_ECC_KEY_BITS BN_P256_KEY_SIZE
#  elif TPM_ECC_SM2_P256
#    define MAX_ECC_KEY_BITS SM2_P256_KEY_SIZE
#  elif TPM_ECC_CURVE_25519
#    define MAX_ECC_KEY_BITS CURVE_25519_KEY_SIZE
#  elif TPM_ECC_BP_P256_R1
#    define MAX_ECC_KEY_BITS BP_P256_R1_KEY_SIZE
// 224
#  elif ECC_NIST_P224
#    define MAX_ECC_KEY_BITS NIST_P224_KEY_SIZE
// 192
#  elif ECC_NIST_P192
#    define MAX_ECC_KEY_BITS NIST_P192_KEY_SIZE
#  else
#    error ALG_ECC enabled, but no ECC Curves Enabled
#  endif

#  define MAX_ECC_KEY_BYTES ((MAX_ECC_KEY_BITS + 7) / 8)

#endif  // ALG_ECC

#if ALG_RSA
// Table "Defines for RSA Asymmetric Cipher Algorithm Constants" (TCG Algorithm
Registry)
#  define RSA_KEY_SIZES_BITS                                                    \
     (RSA_1024 * 1024), (RSA_2048 * 2048), (RSA_3072 * 3072), (RSA_4096 * 4096), \
         (RSA_16384 * 16384)

#  if RSA_16384
#    define RSA_MAX_KEY_SIZE_BITS 16384
```

```
#  elif RSA_4096
#    define RSA_MAX_KEY_SIZE_BITS 4096
#  elif RSA_3072
#    define RSA_MAX_KEY_SIZE_BITS 3072
#  elif RSA_2048
#    define RSA_MAX_KEY_SIZE_BITS 2048
#  elif RSA_1024
#    define RSA_MAX_KEY_SIZE_BITS 1024
#  else
#    error RSA Enabled, but no RSA key sizes enabled.
#  endif

#  define MAX_RSA_KEY_BITS  RSA_MAX_KEY_SIZE_BITS
#  define MAX_RSA_KEY_BYTES BITS_TO_BYTES(RSA_MAX_KEY_SIZE_BITS)
#endif  // ALG_RSA

// Table "Defines for AES Symmetric Cipher Algorithm Constants" (TCG Algorithm
Registry)
#define AES_KEY_SIZES_BITS (AES_128 * 128), (AES_192 * 192), (AES_256 * 256)
#define AES_MAX_KEY_SIZE_BITS \
    MAX((AES_256 * 256), MAX((AES_192 * 192), (AES_128 * 128)))
#define MAX_AES_KEY_BITS  AES_MAX_KEY_SIZE_BITS
#define MAX_AES_KEY_BYTES BITS_TO_BYTES(MAX_AES_KEY_BITS)
#define AES_BLOCK_SIZES   (AES_128 * 128 / 8), (AES_192 * 128 / 8), (AES_256 * 128 /
8)
#define MAX_AES_BLOCK_SIZE_BYTES \
    MAX((AES_256 * 128 / 8), MAX((AES_192 * 128 / 8), (AES_128 * 128 / 8)))
#define AES_MAX_BLOCK_SIZE MAX_AES_BLOCK_SIZE_BYTES

// Table "Defines for SM4 Symmetric Cipher Algorithm Constants" (TCG Algorithm
Registry)
#define SM4_KEY_SIZES_BITS       (SM4_128 * 128)
#define SM4_MAX_KEY_SIZE_BITS    (SM4_128 * 128)
#define MAX_SM4_KEY_BITS         SM4_MAX_KEY_SIZE_BITS
#define MAX_SM4_KEY_BYTES        BITS_TO_BYTES(MAX_SM4_KEY_BITS)
#define SM4_BLOCK_SIZES          (SM4_128 * 128 / 8)
#define MAX_SM4_BLOCK_SIZE_BYTES (SM4_128 * 128 / 8)
#define SM4_MAX_BLOCK_SIZE       MAX_SM4_BLOCK_SIZE_BYTES

// Table "Defines for CAMELLIA Symmetric Cipher Algorithm Constants" (TCG Algorithm
Registry)
#define CAMELLIA_KEY_SIZES_BITS \
    (CAMELLIA_128 * 128), (CAMELLIA_192 * 192), (CAMELLIA_256 * 256)
#define CAMELLIA_MAX_KEY_SIZE_BITS \
    MAX((CAMELLIA_256 * 256), MAX((CAMELLIA_192 * 192), (CAMELLIA_128 * 128)))
#define MAX_CAMELLIA_KEY_BITS  CAMELLIA_MAX_KEY_SIZE_BITS
#define MAX_CAMELLIA_KEY_BYTES BITS_TO_BYTES(MAX_CAMELLIA_KEY_BITS)
#define CAMELLIA_BLOCK_SIZES \
    (CAMELLIA_128 * 128 / 8), (CAMELLIA_192 * 128 / 8), (CAMELLIA_256 * 128 / 8)
#define MAX_CAMELLIA_BLOCK_SIZE_BYTES \
    MAX((CAMELLIA_256 * 128 / 8),     \
        MAX((CAMELLIA_192 * 128 / 8), (CAMELLIA_128 * 128 / 8)))
#define CAMELLIA_MAX_BLOCK_SIZE MAX_CAMELLIA_BLOCK_SIZE_BYTES

// Derived Symmetric Values
#define SYM_COUNT ALG_AES + ALG_SM4 + ALG_CAMELLIA
#define MAX_SYM_BLOCK_SIZE \
    MAX(CAMELLIA_MAX_BLOCK_SIZE, MAX(SM4_MAX_BLOCK_SIZE, AES_MAX_BLOCK_SIZE))
#define MAX_SYM_KEY_BITS \
    MAX(CAMELLIA_MAX_KEY_SIZE_BITS, MAX(SM4_MAX_KEY_SIZE_BITS, AES_MAX_KEY_SIZE_BITS))
#define MAX_SYM_KEY_BYTES ((MAX_SYM_KEY_BITS + 7) / 8)

// Table "Defines for SHA1 Hash Values" (TCG Algorithm Registry)
#define SHA1_DIGEST_SIZE 20
#define SHA1_BLOCK_SIZE  64
```

```
// Table "Defines for SHA256 Hash Values" (TCG Algorithm Registry)
#define SHA256_DIGEST_SIZE 32
#define SHA256_BLOCK_SIZE  64

// Table "Defines for SHA384 Hash Values" (TCG Algorithm Registry)
#define SHA384_DIGEST_SIZE 48
#define SHA384_BLOCK_SIZE  128

// Table "Defines for SHA512 Hash Values" (TCG Algorithm Registry)
#define SHA512_DIGEST_SIZE 64
#define SHA512_BLOCK_SIZE  128

// Table "Defines for SM3_256 Hash Values" (TCG Algorithm Registry)
#define SM3_256_DIGEST_SIZE 32
#define SM3_256_BLOCK_SIZE  64

// Table "Defines for SHA3_256 Hash Values" (TCG Algorithm Registry)
#define SHA3_256_DIGEST_SIZE 32
#define SHA3_256_BLOCK_SIZE  136

// Table "Defines for SHA3_384 Hash Values" (TCG Algorithm Registry)
#define SHA3_384_DIGEST_SIZE 48
#define SHA3_384_BLOCK_SIZE  104

// Table "Defines for SHA3_512 Hash Values" (TCG Algorithm Registry)
#define SHA3_512_DIGEST_SIZE 64
#define SHA3_512_BLOCK_SIZE  72

// Derived Hash Values
#define HASH_COUNT                                                              \
    (ALG_SHA1 + ALG_SHA256 + ALG_SHA384 + ALG_SHA512 + ALG_SM3_256 + ALG_SHA3_256 \
     + ALG_SHA3_384 + ALG_SHA3_512)

// Leaving these as MAX-based calculations because (a) they don't slow down the
// build noticably, and (b) hash block and digest sizes vary, so the #if
// cascades for these are significantly more error prone to maintain.
#define MAX_HASH_BLOCK_SIZE                                         \
    MAX((ALG_SHA3_512 * SHA3_512_BLOCK_SIZE),                       \
        MAX((ALG_SHA3_384 * SHA3_384_BLOCK_SIZE),                   \
            MAX((ALG_SHA3_256 * SHA3_256_BLOCK_SIZE),               \
                MAX((ALG_SM3_256 * SM3_256_BLOCK_SIZE),             \
                    MAX((ALG_SHA512 * SHA512_BLOCK_SIZE),           \
                        MAX((ALG_SHA384 * SHA384_BLOCK_SIZE),       \
                            MAX((ALG_SHA256 * SHA256_BLOCK_SIZE),   \
                                (ALG_SHA1 * SHA1_BLOCK_SIZE))))))))

#define MAX_HASH_DIGEST_SIZE                                         \
    MAX((ALG_SHA3_512 * SHA3_512_DIGEST_SIZE),                       \
        MAX((ALG_SHA3_384 * SHA3_384_DIGEST_SIZE),                   \
            MAX((ALG_SHA3_256 * SHA3_256_DIGEST_SIZE),               \
                MAX((ALG_SM3_256 * SM3_256_DIGEST_SIZE),             \
                    MAX((ALG_SHA512 * SHA512_DIGEST_SIZE),           \
                        MAX((ALG_SHA384 * SHA384_DIGEST_SIZE),       \
                            MAX((ALG_SHA256 * SHA256_DIGEST_SIZE),   \
                                (ALG_SHA1 * SHA1_DIGEST_SIZE))))))))

#define MAX_DIGEST_SIZE MAX_HASH_DIGEST_SIZE

#if MAX_HASH_DIGEST_SIZE == 0 || MAX_HASH_BLOCK_SIZE == 0
#  error "Hash data not valid"
#endif

// Define the 2B structure that would hold any hash block
TPM2B_TYPE(MAX_HASH_BLOCK, MAX_HASH_BLOCK_SIZE);

// Following typedef is for some old code
```

```
typedef TPM2B_MAX_HASH_BLOCK TPM2B_HASH_BLOCK;

#endif  // _TPM_INCLUDE_PRIVATE_TPMALGORITHMDEFINES_H_
```

6.245 /tpm/include/public/TPMB.h

```
//
// This file contains extra TPM2B structures
//

#ifndef _TPMB_H
#define _TPMB_H

//*** Size Types
// These types are used to differentiate the two different size values used.
//
// NUMBYTES is used when a size is a number of bytes (usually a TPM2B)
typedef UINT16 NUMBYTES;

// TPM2B Types
typedef struct
{
    NUMBYTES size;
    BYTE     buffer[1];
} TPM2B, *P2B;
typedef const TPM2B* PC2B;

// This macro helps avoid having to type in the structure in order to create
// a new TPM2B type that is used in a function.
#define TPM2B_TYPE(name, bytes)       \
    typedef union                     \
    {                                 \
        struct                        \
        {                             \
            NUMBYTES size;            \
            BYTE     buffer[(bytes)]; \
        } t;                          \
        TPM2B b;                      \
    } TPM2B_##name

// This macro defines a TPM2B with a constant character value. This macro
// sets the size of the string to the size minus the terminating zero byte.
// This lets the user of the label add their terminating 0. This method
// is chosen so that existing code that provides a label will continue
// to work correctly.

// Macro to instance and initialize a TPM2B value
#define TPM2B_INIT(TYPE, name) TPM2B_##TYPE name = {sizeof(name.t.buffer), {0}}

#define TPM2B_BYTE_VALUE(bytes) TPM2B_TYPE(bytes##_BYTE_VALUE, bytes)

#endif
```

6.246 /tpm/include/public/TpmCalculatedAttributes.h

```
#ifndef _TPM_CALCULATED_ATTRIBUTES_H_
#define _TPM_CALCULATED_ATTRIBUTES_H_

#include "public/TpmAlgorithmDefines.h"
#include "public/GpMacros.h"

#define JOIN(x, y)        x##y
#define JOIN3(x, y, z)    x##y##z
#define CONCAT(x, y)      JOIN(x, y)
```

```
#define CONCAT3(x, y, z) JOIN3(x, y, z)

//** Derived from Vendor-specific values
// Values derived from vendor specific settings in TpmProfile.h
#define PCR_SELECT_MIN    ((PLATFORM_PCR + 7) / 8)
#define PCR_SELECT_MAX    ((IMPLEMENTATION_PCR + 7) / 8)
#define MAX_ORDERLY_COUNT ((1 << ORDERLY_BITS) - 1)
#define RSA_MAX_PRIME     (MAX_RSA_KEY_BYTES / 2)
#define RSA_PRIVATE_SIZE  (RSA_MAX_PRIME * 5)

// If CONTEXT_INTEGRITY_HASH_ALG is defined, then the vendor is using the old style
// table. Otherwise, pick the "strongest" implemented hash algorithm as the context
// hash.
#ifndef CONTEXT_HASH_ALGORITHM
#  if defined ALG_SHA3_512 && ALG_SHA3_512 == YES
#    define CONTEXT_HASH_ALGORITHM SHA3_512
#  elif defined ALG_SHA512 && ALG_SHA512 == YES
#    define CONTEXT_HASH_ALGORITHM SHA512
#  elif defined ALG_SHA3_384 && ALG_SHA3_384 == YES
#    define CONTEXT_HASH_ALGORITHM SHA3_384
#  elif defined ALG_SHA384 && ALG_SHA384 == YES
#    define CONTEXT_HASH_ALGORITHM SHA384
#  elif defined ALG_SHA3_256 && ALG_SHA3_256 == YES
#    define CONTEXT_HASH_ALGORITHM SHA3_256
#  elif defined ALG_SHA256 && ALG_SHA256 == YES
#    define CONTEXT_HASH_ALGORITHM SHA256
#  elif defined ALG_SM3_256 && ALG_SM3_256 == YES
#    define CONTEXT_HASH_ALGORITHM SM3_256
#  elif defined ALG_SHA1 && ALG_SHA1 == YES
#    define CONTEXT_HASH_ALGORITHM SHA1
#  endif
#  define CONTEXT_INTEGRITY_HASH_ALG CONCAT(TPM_ALG_, CONTEXT_HASH_ALGORITHM)
#endif

#ifndef CONTEXT_INTEGRITY_HASH_SIZE
#  define CONTEXT_INTEGRITY_HASH_SIZE CONCAT(CONTEXT_HASH_ALGORITHM, _DIGEST_SIZE)
#endif

#if ALG_RSA
// This table taken from SP800-57 part 1, Table 2.
// for other key lengths, https://csrc.nist.gov/csrc/media/projects/cryptographic-
module-validation-program/documents/fips140-2/fips1402ig.pdf
// provides the following formula for RSA for a key of modulus length L.
// $$x = \frac{1.923 * \sqrt[3]{L * \ln(2)} * \sqrt[3]{(\ln(L*\ln(2)))^2} -
4.69}{\ln(2)}$$
#  define RSA_SECURITY_STRENGTH                        \
      (MAX_RSA_KEY_BITS >= 15360                       \
           ? 256                                       \
           : (MAX_RSA_KEY_BITS >= 7680                 \
                ? 192                                  \
                : (MAX_RSA_KEY_BITS >= 3072            \
                     ? 128                             \
                     : (MAX_RSA_KEY_BITS >= 2048       \
                          ? 112                        \
                          : (MAX_RSA_KEY_BITS >= 1024 ? 80 : 0)))))
#else
#  define RSA_SECURITY_STRENGTH 0
#endif  // ALG_RSA

#if ALG_ECC
#  define ECC_SECURITY_STRENGTH \
      (MAX_ECC_KEY_BITS >= 521  \
           ? 256                \
           : (MAX_ECC_KEY_BITS >= 384 ? 192 : (MAX_ECC_KEY_BITS >= 256 ? 128 : 0)))
#else
#  define ECC_SECURITY_STRENGTH 0
```

```
#endif  // ALG_ECC

#define MAX_ASYM_SECURITY_STRENGTH MAX(RSA_SECURITY_STRENGTH, ECC_SECURITY_STRENGTH)

#define MAX_HASH_SECURITY_STRENGTH ((CONTEXT_INTEGRITY_HASH_SIZE * 8) / 2)

// Unless some algorithm is broken...
#define MAX_SYM_SECURITY_STRENGTH MAX_SYM_KEY_BITS

#define MAX_SECURITY_STRENGTH_BITS  \
    MAX(MAX_ASYM_SECURITY_STRENGTH, \
        MAX(MAX_SYM_SECURITY_STRENGTH, MAX_HASH_SECURITY_STRENGTH))

// This is the size that was used before the 1.38 errata requiring that P1.14.4 be
// followed
#define PROOF_SIZE CONTEXT_INTEGRITY_HASH_SIZE

// As required by P1.14.4
#define COMPLIANT_PROOF_SIZE \
    (MAX(CONTEXT_INTEGRITY_HASH_SIZE, (2 * MAX_SYM_KEY_BYTES)))

// As required by P1.14.3.1
#define COMPLIANT_PRIMARY_SEED_SIZE BITS_TO_BYTES(MAX_SECURITY_STRENGTH_BITS * 2)

// This is the pre-errata version
#ifndef PRIMARY_SEED_SIZE
#  define PRIMARY_SEED_SIZE PROOF_SIZE
#endif

#if USE_SPEC_COMPLIANT_PROOFS
#  undef PROOF_SIZE
#  define PROOF_SIZE COMPLIANT_PROOF_SIZE
#  undef PRIMARY_SEED_SIZE
#  define PRIMARY_SEED_SIZE COMPLIANT_PRIMARY_SEED_SIZE
#endif  // USE_SPEC_COMPLIANT_PROOFS

#if !SKIP_PROOF_ERRORS
#  if PROOF_SIZE < COMPLIANT_PROOF_SIZE
#    error "PROOF_SIZE is not compliant with TPM specification"
#  endif
#  if PRIMARY_SEED_SIZE < COMPLIANT_PRIMARY_SEED_SIZE
#    error Non-compliant PRIMARY_SEED_SIZE
#  endif
#endif  // !SKIP_PROOF_ERRORS

// If CONTEXT_ENCRYPT_ALG is defined, then the vendor is using the old style table
#if defined CONTEXT_ENCRYPT_ALG
#  undef CONTEXT_ENCRYPT_ALGORITHM
#  if CONTEXT_ENCRYPT_ALG == ALG_AES_VALUE
#    define CONTEXT_ENCRYPT_ALGORITHM AES
#  elif CONTEXT_ENCRYPT_ALG == ALG_SM4_VALUE
#    define CONTEXT_ENCRYPT_ALGORITHM SM4
#  elif CONTEXT_ENCRYPT_ALG == ALG_CAMELLIA_VALUE
#    define CONTEXT_ENCRYPT_ALGORITHM CAMELLIA
#  else
#    error Unknown value for CONTEXT_ENCRYPT_ALG
#  endif  // CONTEXT_ENCRYPT_ALG == ALG_AES_VALUE
#else
#  define CONTEXT_ENCRYPT_ALG CONCAT3(ALG_, CONTEXT_ENCRYPT_ALGORITHM, _VALUE)
#endif  // CONTEXT_ENCRYPT_ALG
#define CONTEXT_ENCRYPT_KEY_BITS  CONCAT(CONTEXT_ENCRYPT_ALGORITHM,
_MAX_KEY_SIZE_BITS)
#define CONTEXT_ENCRYPT_KEY_BYTES ((CONTEXT_ENCRYPT_KEY_BITS + 7) / 8)

// This is updated to follow the requirement of P2 that the label not be larger
// than 32 bytes.
```

```
#ifndef LABEL_MAX_BUFFER
#  define LABEL_MAX_BUFFER MIN(32, MAX(MAX_ECC_KEY_BYTES, MAX_DIGEST_SIZE))
#endif

// This bit is used to indicate that an authorization ticket expires on TPM Reset
// and TPM Restart. It is added to the timeout value returned by TPM2_PoliySigned()
// and TPM2_PolicySecret() and used by TPM2_PolicyTicket(). The timeout value is
// relative to Time (g_time). Time is reset whenever the TPM loses power and cannot
// be moved forward by the user (as can Clock). 'g_time' is a 64-bit value expressing
// time in ms. Stealing the MSb for a flag means that the TPM needs to be reset
// at least once every 292,471,208 years rather than once every 584,942,417 years.
#define EXPIRATION_BIT ((UINT64)1 << 63)

// This definition is moved from TpmProfile.h because it is not actually vendor-
// specific. It has to be the same size as the 'sequence' parameter of a TPMS_CONTEXT
// and that is a UINT64. So, this is an invariant value
#define CONTEXT_COUNTER UINT64
#endif  // _TPM_CALCULATED_ATTRIBUTES_H_
```

6.247 /tpm/include/public/TpmTypes.h

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#ifndef _TPM_INCLUDE_PRIVATE_TPMTYPES_H_
#define _TPM_INCLUDE_PRIVATE_TPMTYPES_H_

#ifndef MAX_CAP_BUFFER
#  error MAX_CAP_BUFFER must be defined before this file so it can calculate maximum
capability sizes
#endif
#include "public/Capabilities.h"
#include "public/TpmAlgorithmDefines.h"
#include "public/TpmCalculatedAttributes.h"
#include "public/GpMacros.h"

// Table "Definition of Types for Documentation Clarity" (Part 2: Structures)
typedef UINT32 TPM_ALGORITHM_ID;
#define TYPE_OF_TPM_ALGORITHM_ID UINT32
typedef UINT32 TPM_MODIFIER_INDICATOR;
#define TYPE_OF_TPM_MODIFIER_INDICATOR UINT32
typedef UINT32 TPM_AUTHORIZATION_SIZE;
#define TYPE_OF_TPM_AUTHORIZATION_SIZE UINT32
typedef UINT32 TPM_PARAMETER_SIZE;
#define TYPE_OF_TPM_PARAMETER_SIZE UINT32
typedef UINT16 TPM_KEY_SIZE;
#define TYPE_OF_TPM_KEY_SIZE UINT16
typedef UINT16 TPM_KEY_BITS;
#define TYPE_OF_TPM_KEY_BITS UINT16

// Table "Definition of TPM_CONSTANTS32 Constants" (Part 2: Structures)
typedef UINT32 TPM_CONSTANTS32;
#define TYPE_OF_TPM_CONSTANTS32 UINT32
#define TPM_GENERATED_VALUE     (TPM_CONSTANTS32)(0xFF544347)
#define TPM_MAX_DERIVATION_BITS (TPM_CONSTANTS32)(8192)

// Table "Definition of TPM_ALG_ID Constants" (Part 2: Structures)
typedef UINT16 TPM_ALG_ID;
#define TYPE_OF_TPM_ALG_ID       UINT16
#define ALG_ERROR_VALUE          0x0000
#define TPM_ALG_ERROR            (TPM_ALG_ID)(ALG_ERROR_VALUE)
#define ALG_RSA_VALUE            0x0001
#define TPM_ALG_RSA              (TPM_ALG_ID)(ALG_RSA_VALUE)
#define ALG_TDES_VALUE           0x0003
#define TPM_ALG_TDES             (TPM_ALG_ID)(ALG_TDES_VALUE)
#define ALG_SHA1_VALUE           0x0004
```

```
#define TPM_ALG_SHA1              (TPM_ALG_ID)(ALG_SHA1_VALUE)
#define ALG_HMAC_VALUE            0x0005
#define TPM_ALG_HMAC              (TPM_ALG_ID)(ALG_HMAC_VALUE)
#define ALG_AES_VALUE             0x0006
#define TPM_ALG_AES               (TPM_ALG_ID)(ALG_AES_VALUE)
#define ALG_MGF1_VALUE            0x0007
#define TPM_ALG_MGF1              (TPM_ALG_ID)(ALG_MGF1_VALUE)
#define ALG_KEYEDHASH_VALUE       0x0008
#define TPM_ALG_KEYEDHASH         (TPM_ALG_ID)(ALG_KEYEDHASH_VALUE)
#define ALG_XOR_VALUE             0x000A
#define TPM_ALG_XOR               (TPM_ALG_ID)(ALG_XOR_VALUE)
#define ALG_SHA256_VALUE          0x000B
#define TPM_ALG_SHA256            (TPM_ALG_ID)(ALG_SHA256_VALUE)
#define ALG_SHA384_VALUE          0x000C
#define TPM_ALG_SHA384            (TPM_ALG_ID)(ALG_SHA384_VALUE)
#define ALG_SHA512_VALUE          0x000D
#define TPM_ALG_SHA512            (TPM_ALG_ID)(ALG_SHA512_VALUE)
#define ALG_SHA256_192_VALUE      0x000E
#define TPM_ALG_SHA256_192        (TPM_ALG_ID)(ALG_SHA256_192_VALUE)
#define ALG_NULL_VALUE            0x0010
#define TPM_ALG_NULL              (TPM_ALG_ID)(ALG_NULL_VALUE)
#define ALG_SM3_256_VALUE         0x0012
#define TPM_ALG_SM3_256           (TPM_ALG_ID)(ALG_SM3_256_VALUE)
#define ALG_SM4_VALUE             0x0013
#define TPM_ALG_SM4               (TPM_ALG_ID)(ALG_SM4_VALUE)
#define ALG_RSASSA_VALUE          0x0014
#define TPM_ALG_RSASSA            (TPM_ALG_ID)(ALG_RSASSA_VALUE)
#define ALG_RSAES_VALUE           0x0015
#define TPM_ALG_RSAES             (TPM_ALG_ID)(ALG_RSAES_VALUE)
#define ALG_RSAPSS_VALUE          0x0016
#define TPM_ALG_RSAPSS            (TPM_ALG_ID)(ALG_RSAPSS_VALUE)
#define ALG_OAEP_VALUE            0x0017
#define TPM_ALG_OAEP              (TPM_ALG_ID)(ALG_OAEP_VALUE)
#define ALG_ECDSA_VALUE           0x0018
#define TPM_ALG_ECDSA             (TPM_ALG_ID)(ALG_ECDSA_VALUE)
#define ALG_ECDH_VALUE            0x0019
#define TPM_ALG_ECDH              (TPM_ALG_ID)(ALG_ECDH_VALUE)
#define ALG_ECDAA_VALUE           0x001A
#define TPM_ALG_ECDAA             (TPM_ALG_ID)(ALG_ECDAA_VALUE)
#define ALG_SM2_VALUE             0x001B
#define TPM_ALG_SM2               (TPM_ALG_ID)(ALG_SM2_VALUE)
#define ALG_ECSCHNORR_VALUE       0x001C
#define TPM_ALG_ECSCHNORR         (TPM_ALG_ID)(ALG_ECSCHNORR_VALUE)
#define ALG_ECMQV_VALUE           0x001D
#define TPM_ALG_ECMQV             (TPM_ALG_ID)(ALG_ECMQV_VALUE)
#define ALG_KDF1_SP800_56A_VALUE  0x0020
#define TPM_ALG_KDF1_SP800_56A    (TPM_ALG_ID)(ALG_KDF1_SP800_56A_VALUE)
#define ALG_KDF2_VALUE            0x0021
#define TPM_ALG_KDF2              (TPM_ALG_ID)(ALG_KDF2_VALUE)
#define ALG_KDF1_SP800_108_VALUE  0x0022
#define TPM_ALG_KDF1_SP800_108    (TPM_ALG_ID)(ALG_KDF1_SP800_108_VALUE)
#define ALG_ECC_VALUE             0x0023
#define TPM_ALG_ECC               (TPM_ALG_ID)(ALG_ECC_VALUE)
#define ALG_SYMCIPHER_VALUE       0x0025
#define TPM_ALG_SYMCIPHER         (TPM_ALG_ID)(ALG_SYMCIPHER_VALUE)
#define ALG_CAMELLIA_VALUE        0x0026
#define TPM_ALG_CAMELLIA          (TPM_ALG_ID)(ALG_CAMELLIA_VALUE)
#define ALG_SHA3_256_VALUE        0x0027
#define TPM_ALG_SHA3_256          (TPM_ALG_ID)(ALG_SHA3_256_VALUE)
#define ALG_SHA3_384_VALUE        0x0028
#define TPM_ALG_SHA3_384          (TPM_ALG_ID)(ALG_SHA3_384_VALUE)
#define ALG_SHA3_512_VALUE        0x0029
#define TPM_ALG_SHA3_512          (TPM_ALG_ID)(ALG_SHA3_512_VALUE)
#define ALG_SHAKE128_VALUE        0x002A
#define TPM_ALG_SHAKE128          (TPM_ALG_ID)(ALG_SHAKE128_VALUE)
#define ALG_SHAKE256_VALUE        0x002B
```

```
#define TPM_ALG_SHAKE256          (TPM_ALG_ID)(ALG_SHAKE256_VALUE)
#define ALG_SHAKE256_192_VALUE    0x002C
#define TPM_ALG_SHAKE256_192      (TPM_ALG_ID)(ALG_SHAKE256_192_VALUE)
#define ALG_SHAKE256_256_VALUE    0x002D
#define TPM_ALG_SHAKE256_256      (TPM_ALG_ID)(ALG_SHAKE256_256_VALUE)
#define ALG_SHAKE256_512_VALUE    0x002E
#define TPM_ALG_SHAKE256_512      (TPM_ALG_ID)(ALG_SHAKE256_512_VALUE)
#define ALG_CMAC_VALUE            0x003F
#define TPM_ALG_CMAC              (TPM_ALG_ID)(ALG_CMAC_VALUE)
#define ALG_CTR_VALUE             0x0040
#define TPM_ALG_CTR               (TPM_ALG_ID)(ALG_CTR_VALUE)
#define ALG_OFB_VALUE             0x0041
#define TPM_ALG_OFB               (TPM_ALG_ID)(ALG_OFB_VALUE)
#define ALG_CBC_VALUE             0x0042
#define TPM_ALG_CBC               (TPM_ALG_ID)(ALG_CBC_VALUE)
#define ALG_CFB_VALUE             0x0043
#define TPM_ALG_CFB               (TPM_ALG_ID)(ALG_CFB_VALUE)
#define ALG_ECB_VALUE             0x0044
#define TPM_ALG_ECB               (TPM_ALG_ID)(ALG_ECB_VALUE)
#define ALG_CCM_VALUE             0x0050
#define TPM_ALG_CCM               (TPM_ALG_ID)(ALG_CCM_VALUE)
#define ALG_GCM_VALUE             0x0051
#define TPM_ALG_GCM               (TPM_ALG_ID)(ALG_GCM_VALUE)
#define ALG_KW_VALUE              0x0052
#define TPM_ALG_KW                (TPM_ALG_ID)(ALG_KW_VALUE)
#define ALG_KWP_VALUE             0x0053
#define TPM_ALG_KWP               (TPM_ALG_ID)(ALG_KWP_VALUE)
#define ALG_EAX_VALUE             0x0054
#define TPM_ALG_EAX               (TPM_ALG_ID)(ALG_EAX_VALUE)
#define ALG_EDDSA_VALUE           0x0060
#define TPM_ALG_EDDSA             (TPM_ALG_ID)(ALG_EDDSA_VALUE)
#define ALG_EDDSA_PH_VALUE        0x0061
#define TPM_ALG_EDDSA_PH          (TPM_ALG_ID)(ALG_EDDSA_PH_VALUE)
#define ALG_LMS_VALUE             0x0070
#define TPM_ALG_LMS               (TPM_ALG_ID)(ALG_LMS_VALUE)
#define ALG_XMSS_VALUE            0x0071
#define TPM_ALG_XMSS              (TPM_ALG_ID)(ALG_XMSS_VALUE)
//        Values derived from Table "Definition of TPM_ALG_ID Constants" (Part 2:
Structures)
#define ALG_FIRST_VALUE 0x0001
#define TPM_ALG_FIRST    (TPM_ALG_ID)(ALG_FIRST_VALUE)
#define ALG_LAST_VALUE  0x0071
#define TPM_ALG_LAST     (TPM_ALG_ID)(ALG_LAST_VALUE)

// Table "Definition of TPM_ECC_CURVE Constants" (Part 2: Structures)
typedef UINT16 TPM_ECC_CURVE;

#define TYPE_OF_TPM_ECC_CURVE UINT16
#define TPM_ECC_NONE           (TPM_ECC_CURVE)(0x0000)
#define TPM_ECC_NIST_P192      (TPM_ECC_CURVE)(0x0001)
#define TPM_ECC_NIST_P224      (TPM_ECC_CURVE)(0x0002)
#define TPM_ECC_NIST_P256      (TPM_ECC_CURVE)(0x0003)
#define TPM_ECC_NIST_P384      (TPM_ECC_CURVE)(0x0004)
#define TPM_ECC_NIST_P521      (TPM_ECC_CURVE)(0x0005)
#define TPM_ECC_BN_P256        (TPM_ECC_CURVE)(0x0010)
#define TPM_ECC_BN_P638        (TPM_ECC_CURVE)(0x0011)
#define TPM_ECC_SM2_P256       (TPM_ECC_CURVE)(0x0020)
#define TPM_ECC_BP_P256_R1     (TPM_ECC_CURVE)(0x0030)
#define TPM_ECC_BP_P384_R1     (TPM_ECC_CURVE)(0x0031)
#define TPM_ECC_BP_P512_R1     (TPM_ECC_CURVE)(0x0032)
#define TPM_ECC_CURVE_25519    (TPM_ECC_CURVE)(0x0040)
#define TPM_ECC_CURVE_448      (TPM_ECC_CURVE)(0x0041)

// Table "Definition of TPM_CC Constants" (Part 2: Structures)
typedef UINT32 TPM_CC;
```

```
#define TYPE_OF_TPM_CC                    UINT32
#define TPM_CC_FIRST                      (TPM_CC)(0x0000011F)
#define TPM_CC_NV_UndefineSpaceSpecial    (TPM_CC)(0x0000011F)
#define TPM_CC_EvictControl               (TPM_CC)(0x00000120)
#define TPM_CC_HierarchyControl           (TPM_CC)(0x00000121)
#define TPM_CC_NV_UndefineSpace           (TPM_CC)(0x00000122)
#define TPM_CC_ChangeEPS                  (TPM_CC)(0x00000124)
#define TPM_CC_ChangePPS                  (TPM_CC)(0x00000125)
#define TPM_CC_Clear                      (TPM_CC)(0x00000126)
#define TPM_CC_ClearControl               (TPM_CC)(0x00000127)
#define TPM_CC_ClockSet                   (TPM_CC)(0x00000128)
#define TPM_CC_HierarchyChangeAuth        (TPM_CC)(0x00000129)
#define TPM_CC_NV_DefineSpace             (TPM_CC)(0x0000012A)
#define TPM_CC_PCR_Allocate               (TPM_CC)(0x0000012B)
#define TPM_CC_PCR_SetAuthPolicy          (TPM_CC)(0x0000012C)
#define TPM_CC_PP_Commands                (TPM_CC)(0x0000012D)
#define TPM_CC_SetPrimaryPolicy           (TPM_CC)(0x0000012E)
#define TPM_CC_FieldUpgradeStart          (TPM_CC)(0x0000012F)
#define TPM_CC_ClockRateAdjust            (TPM_CC)(0x00000130)
#define TPM_CC_CreatePrimary              (TPM_CC)(0x00000131)
#define TPM_CC_NV_GlobalWriteLock         (TPM_CC)(0x00000132)
#define TPM_CC_GetCommandAuditDigest      (TPM_CC)(0x00000133)
#define TPM_CC_NV_Increment               (TPM_CC)(0x00000134)
#define TPM_CC_NV_SetBits                 (TPM_CC)(0x00000135)
#define TPM_CC_NV_Extend                  (TPM_CC)(0x00000136)
#define TPM_CC_NV_Write                   (TPM_CC)(0x00000137)
#define TPM_CC_NV_WriteLock               (TPM_CC)(0x00000138)
#define TPM_CC_DictionaryAttackLockReset  (TPM_CC)(0x00000139)
#define TPM_CC_DictionaryAttackParameters (TPM_CC)(0x0000013A)
#define TPM_CC_NV_ChangeAuth              (TPM_CC)(0x0000013B)
#define TPM_CC_PCR_Event                  (TPM_CC)(0x0000013C)
#define TPM_CC_PCR_Reset                  (TPM_CC)(0x0000013D)
#define TPM_CC_SequenceComplete           (TPM_CC)(0x0000013E)
#define TPM_CC_SetAlgorithmSet            (TPM_CC)(0x0000013F)
#define TPM_CC_SetCommandCodeAuditStatus  (TPM_CC)(0x00000140)
#define TPM_CC_FieldUpgradeData           (TPM_CC)(0x00000141)
#define TPM_CC_IncrementalSelfTest        (TPM_CC)(0x00000142)
#define TPM_CC_SelfTest                   (TPM_CC)(0x00000143)
#define TPM_CC_Startup                    (TPM_CC)(0x00000144)
#define TPM_CC_Shutdown                   (TPM_CC)(0x00000145)
#define TPM_CC_StirRandom                 (TPM_CC)(0x00000146)
#define TPM_CC_ActivateCredential         (TPM_CC)(0x00000147)
#define TPM_CC_Certify                    (TPM_CC)(0x00000148)
#define TPM_CC_PolicyNV                   (TPM_CC)(0x00000149)
#define TPM_CC_CertifyCreation            (TPM_CC)(0x0000014A)
#define TPM_CC_Duplicate                  (TPM_CC)(0x0000014B)
#define TPM_CC_GetTime                    (TPM_CC)(0x0000014C)
#define TPM_CC_GetSessionAuditDigest      (TPM_CC)(0x0000014D)
#define TPM_CC_NV_Read                    (TPM_CC)(0x0000014E)
#define TPM_CC_NV_ReadLock                (TPM_CC)(0x0000014F)
#define TPM_CC_ObjectChangeAuth           (TPM_CC)(0x00000150)
#define TPM_CC_PolicySecret               (TPM_CC)(0x00000151)
#define TPM_CC_Rewrap                     (TPM_CC)(0x00000152)
#define TPM_CC_Create                     (TPM_CC)(0x00000153)
#define TPM_CC_ECDH_ZGen                  (TPM_CC)(0x00000154)
#define TPM_CC_HMAC                       (TPM_CC)(0x00000155)
#define TPM_CC_MAC                        (TPM_CC)(0x00000155)
#define TPM_CC_Import                     (TPM_CC)(0x00000156)
#define TPM_CC_Load                       (TPM_CC)(0x00000157)
#define TPM_CC_Quote                      (TPM_CC)(0x00000158)
#define TPM_CC_RSA_Decrypt                (TPM_CC)(0x00000159)
#define TPM_CC_HMAC_Start                 (TPM_CC)(0x0000015B)
#define TPM_CC_MAC_Start                  (TPM_CC)(0x0000015B)
#define TPM_CC_SequenceUpdate             (TPM_CC)(0x0000015C)
#define TPM_CC_Sign                       (TPM_CC)(0x0000015D)
#define TPM_CC_Unseal                     (TPM_CC)(0x0000015E)
```

```
#define TPM_CC_PolicySigned                (TPM_CC)(0x00000160)
#define TPM_CC_ContextLoad                 (TPM_CC)(0x00000161)
#define TPM_CC_ContextSave                 (TPM_CC)(0x00000162)
#define TPM_CC_ECDH_KeyGen                 (TPM_CC)(0x00000163)
#define TPM_CC_EncryptDecrypt              (TPM_CC)(0x00000164)
#define TPM_CC_FlushContext                (TPM_CC)(0x00000165)
#define TPM_CC_LoadExternal                (TPM_CC)(0x00000167)
#define TPM_CC_MakeCredential              (TPM_CC)(0x00000168)
#define TPM_CC_NV_ReadPublic               (TPM_CC)(0x00000169)
#define TPM_CC_PolicyAuthorize             (TPM_CC)(0x0000016A)
#define TPM_CC_PolicyAuthValue             (TPM_CC)(0x0000016B)
#define TPM_CC_PolicyCommandCode           (TPM_CC)(0x0000016C)
#define TPM_CC_PolicyCounterTimer          (TPM_CC)(0x0000016D)
#define TPM_CC_PolicyCpHash                (TPM_CC)(0x0000016E)
#define TPM_CC_PolicyLocality              (TPM_CC)(0x0000016F)
#define TPM_CC_PolicyNameHash              (TPM_CC)(0x00000170)
#define TPM_CC_PolicyOR                    (TPM_CC)(0x00000171)
#define TPM_CC_PolicyTicket                (TPM_CC)(0x00000172)
#define TPM_CC_ReadPublic                  (TPM_CC)(0x00000173)
#define TPM_CC_RSA_Encrypt                 (TPM_CC)(0x00000174)
#define TPM_CC_StartAuthSession            (TPM_CC)(0x00000176)
#define TPM_CC_VerifySignature             (TPM_CC)(0x00000177)
#define TPM_CC_ECC_Parameters              (TPM_CC)(0x00000178)
#define TPM_CC_FirmwareRead                (TPM_CC)(0x00000179)
#define TPM_CC_GetCapability               (TPM_CC)(0x0000017A)
#define TPM_CC_GetRandom                   (TPM_CC)(0x0000017B)
#define TPM_CC_GetTestResult               (TPM_CC)(0x0000017C)
#define TPM_CC_Hash                        (TPM_CC)(0x0000017D)
#define TPM_CC_PCR_Read                    (TPM_CC)(0x0000017E)
#define TPM_CC_PolicyPCR                   (TPM_CC)(0x0000017F)
#define TPM_CC_PolicyRestart               (TPM_CC)(0x00000180)
#define TPM_CC_ReadClock                   (TPM_CC)(0x00000181)
#define TPM_CC_PCR_Extend                  (TPM_CC)(0x00000182)
#define TPM_CC_PCR_SetAuthValue            (TPM_CC)(0x00000183)
#define TPM_CC_NV_Certify                  (TPM_CC)(0x00000184)
#define TPM_CC_EventSequenceComplete       (TPM_CC)(0x00000185)
#define TPM_CC_HashSequenceStart           (TPM_CC)(0x00000186)
#define TPM_CC_PolicyPhysicalPresence      (TPM_CC)(0x00000187)
#define TPM_CC_PolicyDuplicationSelect     (TPM_CC)(0x00000188)
#define TPM_CC_PolicyGetDigest             (TPM_CC)(0x00000189)
#define TPM_CC_TestParms                   (TPM_CC)(0x0000018A)
#define TPM_CC_Commit                      (TPM_CC)(0x0000018B)
#define TPM_CC_PolicyPassword              (TPM_CC)(0x0000018C)
#define TPM_CC_ZGen_2Phase                 (TPM_CC)(0x0000018D)
#define TPM_CC_EC_Ephemeral                (TPM_CC)(0x0000018E)
#define TPM_CC_PolicyNvWritten             (TPM_CC)(0x0000018F)
#define TPM_CC_PolicyTemplate              (TPM_CC)(0x00000190)
#define TPM_CC_CreateLoaded                (TPM_CC)(0x00000191)
#define TPM_CC_PolicyAuthorizeNV           (TPM_CC)(0x00000192)
#define TPM_CC_EncryptDecrypt2             (TPM_CC)(0x00000193)
#define TPM_CC_AC_GetCapability            (TPM_CC)(0x00000194)
#define TPM_CC_AC_Send                     (TPM_CC)(0x00000195)
#define TPM_CC_Policy_AC_SendSelect        (TPM_CC)(0x00000196)
#define TPM_CC_CertifyX509                 (TPM_CC)(0x00000197)
#define TPM_CC_ACT_SetTimeout              (TPM_CC)(0x00000198)
#define TPM_CC_ECC_Encrypt                 (TPM_CC)(0x00000199)
#define TPM_CC_ECC_Decrypt                 (TPM_CC)(0x0000019A)
#define TPM_CC_PolicyCapability            (TPM_CC)(0x0000019B)
#define TPM_CC_PolicyParameters            (TPM_CC)(0x0000019C)
#define TPM_CC_NV_DefineSpace2             (TPM_CC)(0x0000019D)
#define TPM_CC_NV_ReadPublic2              (TPM_CC)(0x0000019E)
#define TPM_CC_SetCapability               (TPM_CC)(0x0000019F)
#define TPM_CC_LAST                        (TPM_CC)(0x0000019F)
#define CC_VEND                            (TPM_CC)(0x20000000)
#define TPM_CC_Vendor_TCG_Test             (TPM_CC)(0x20000000)
```

```
// This large macro is needed to determine the maximum commandIndex. This value
// is needed in order to size typdef'ed structures. As a consequence, the
// computation cannot be deferred until the command array is instanced and
// so that the number of entires can be determine by
// sizeof(array)/sizeof(entry).
//
// Size the array of library commands based on whether or not the array is
// packed (only defined commands) or dense
// (having entries for unimplemented commands). This overly large macro
// computes the size of the array and sets some global constants
#if COMPRESSED_LISTS
#  define ADD_FILL 0
#else
#  define ADD_FILL 1
#endif
#define LIBRARY_COMMAND_ARRAY_SIZE                                          \
    (0 + (ADD_FILL || CC_NV_UndefineSpaceSpecial)     /* 0x0000011F */ \
     + (ADD_FILL || CC_EvictControl)                  /* 0x00000120 */ \
     + (ADD_FILL || CC_HierarchyControl)              /* 0x00000121 */ \
     + (ADD_FILL || CC_NV_UndefineSpace)              /* 0x00000122 */ \
     + ADD_FILL                                       /* 0x00000123 */ \
     + (ADD_FILL || CC_ChangeEPS)                     /* 0x00000124 */ \
     + (ADD_FILL || CC_ChangePPS)                     /* 0x00000125 */ \
     + (ADD_FILL || CC_Clear)                         /* 0x00000126 */ \
     + (ADD_FILL || CC_ClearControl)                  /* 0x00000127 */ \
     + (ADD_FILL || CC_ClockSet)                      /* 0x00000128 */ \
     + (ADD_FILL || CC_HierarchyChangeAuth)           /* 0x00000129 */ \
     + (ADD_FILL || CC_NV_DefineSpace)                /* 0x0000012A */ \
     + (ADD_FILL || CC_PCR_Allocate)                  /* 0x0000012B */ \
     + (ADD_FILL || CC_PCR_SetAuthPolicy)             /* 0x0000012C */ \
     + (ADD_FILL || CC_PP_Commands)                   /* 0x0000012D */ \
     + (ADD_FILL || CC_SetPrimaryPolicy)              /* 0x0000012E */ \
     + (ADD_FILL || CC_FieldUpgradeStart)             /* 0x0000012F */ \
     + (ADD_FILL || CC_ClockRateAdjust)               /* 0x00000130 */ \
     + (ADD_FILL || CC_CreatePrimary)                 /* 0x00000131 */ \
     + (ADD_FILL || CC_NV_GlobalWriteLock)            /* 0x00000132 */ \
     + (ADD_FILL || CC_GetCommandAuditDigest)         /* 0x00000133 */ \
     + (ADD_FILL || CC_NV_Increment)                  /* 0x00000134 */ \
     + (ADD_FILL || CC_NV_SetBits)                    /* 0x00000135 */ \
     + (ADD_FILL || CC_NV_Extend)                     /* 0x00000136 */ \
     + (ADD_FILL || CC_NV_Write)                      /* 0x00000137 */ \
     + (ADD_FILL || CC_NV_WriteLock)                  /* 0x00000138 */ \
     + (ADD_FILL || CC_DictionaryAttackLockReset)     /* 0x00000139 */ \
     + (ADD_FILL || CC_DictionaryAttackParameters)    /* 0x0000013A */ \
     + (ADD_FILL || CC_NV_ChangeAuth)                 /* 0x0000013B */ \
     + (ADD_FILL || CC_PCR_Event)                     /* 0x0000013C */ \
     + (ADD_FILL || CC_PCR_Reset)                     /* 0x0000013D */ \
     + (ADD_FILL || CC_SequenceComplete)              /* 0x0000013E */ \
     + (ADD_FILL || CC_SetAlgorithmSet)               /* 0x0000013F */ \
     + (ADD_FILL || CC_SetCommandCodeAuditStatus)     /* 0x00000140 */ \
     + (ADD_FILL || CC_FieldUpgradeData)              /* 0x00000141 */ \
     + (ADD_FILL || CC_IncrementalSelfTest)           /* 0x00000142 */ \
     + (ADD_FILL || CC_SelfTest)                      /* 0x00000143 */ \
     + (ADD_FILL || CC_Startup)                       /* 0x00000144 */ \
     + (ADD_FILL || CC_Shutdown)                      /* 0x00000145 */ \
     + (ADD_FILL || CC_StirRandom)                    /* 0x00000146 */ \
     + (ADD_FILL || CC_ActivateCredential)            /* 0x00000147 */ \
     + (ADD_FILL || CC_Certify)                       /* 0x00000148 */ \
     + (ADD_FILL || CC_PolicyNV)                      /* 0x00000149 */ \
     + (ADD_FILL || CC_CertifyCreation)               /* 0x0000014A */ \
     + (ADD_FILL || CC_Duplicate)                     /* 0x0000014B */ \
     + (ADD_FILL || CC_GetTime)                       /* 0x0000014C */ \
     + (ADD_FILL || CC_GetSessionAuditDigest)         /* 0x0000014D */ \
     + (ADD_FILL || CC_NV_Read)                       /* 0x0000014E */ \
     + (ADD_FILL || CC_NV_ReadLock)                   /* 0x0000014F */ \
     + (ADD_FILL || CC_ObjectChangeAuth)              /* 0x00000150 */ \
```

```
+ (ADD_FILL || CC_PolicySecret)                 /* 0x00000151 */ \
+ (ADD_FILL || CC_Rewrap)                       /* 0x00000152 */ \
+ (ADD_FILL || CC_Create)                       /* 0x00000153 */ \
+ (ADD_FILL || CC_ECDH_ZGen)                    /* 0x00000154 */ \
+ (ADD_FILL || (CC_HMAC || CC_MAC))             /* 0x00000155 */ \
+ (ADD_FILL || CC_Import)                       /* 0x00000156 */ \
+ (ADD_FILL || CC_Load)                         /* 0x00000157 */ \
+ (ADD_FILL || CC_Quote)                        /* 0x00000158 */ \
+ (ADD_FILL || CC_RSA_Decrypt)                  /* 0x00000159 */ \
+ ADD_FILL                                      /* 0x0000015A */ \
+ (ADD_FILL || (CC_HMAC_Start || CC_MAC_Start)) /* 0x0000015B */ \
+ (ADD_FILL || CC_SequenceUpdate)               /* 0x0000015C */ \
+ (ADD_FILL || CC_Sign)                         /* 0x0000015D */ \
+ (ADD_FILL || CC_Unseal)                       /* 0x0000015E */ \
+ ADD_FILL                                      /* 0x0000015F */ \
+ (ADD_FILL || CC_PolicySigned)                 /* 0x00000160 */ \
+ (ADD_FILL || CC_ContextLoad)                  /* 0x00000161 */ \
+ (ADD_FILL || CC_ContextSave)                  /* 0x00000162 */ \
+ (ADD_FILL || CC_ECDH_KeyGen)                  /* 0x00000163 */ \
+ (ADD_FILL || CC_EncryptDecrypt)               /* 0x00000164 */ \
+ (ADD_FILL || CC_FlushContext)                 /* 0x00000165 */ \
+ ADD_FILL                                      /* 0x00000166 */ \
+ (ADD_FILL || CC_LoadExternal)                 /* 0x00000167 */ \
+ (ADD_FILL || CC_MakeCredential)               /* 0x00000168 */ \
+ (ADD_FILL || CC_NV_ReadPublic)                /* 0x00000169 */ \
+ (ADD_FILL || CC_PolicyAuthorize)              /* 0x0000016A */ \
+ (ADD_FILL || CC_PolicyAuthValue)              /* 0x0000016B */ \
+ (ADD_FILL || CC_PolicyCommandCode)            /* 0x0000016C */ \
+ (ADD_FILL || CC_PolicyCounterTimer)           /* 0x0000016D */ \
+ (ADD_FILL || CC_PolicyCpHash)                 /* 0x0000016E */ \
+ (ADD_FILL || CC_PolicyLocality)               /* 0x0000016F */ \
+ (ADD_FILL || CC_PolicyNameHash)               /* 0x00000170 */ \
+ (ADD_FILL || CC_PolicyOR)                     /* 0x00000171 */ \
+ (ADD_FILL || CC_PolicyTicket)                 /* 0x00000172 */ \
+ (ADD_FILL || CC_ReadPublic)                   /* 0x00000173 */ \
+ (ADD_FILL || CC_RSA_Encrypt)                  /* 0x00000174 */ \
+ ADD_FILL                                      /* 0x00000175 */ \
+ (ADD_FILL || CC_StartAuthSession)             /* 0x00000176 */ \
+ (ADD_FILL || CC_VerifySignature)              /* 0x00000177 */ \
+ (ADD_FILL || CC_ECC_Parameters)               /* 0x00000178 */ \
+ (ADD_FILL || CC_FirmwareRead)                 /* 0x00000179 */ \
+ (ADD_FILL || CC_GetCapability)                /* 0x0000017A */ \
+ (ADD_FILL || CC_GetRandom)                    /* 0x0000017B */ \
+ (ADD_FILL || CC_GetTestResult)                /* 0x0000017C */ \
+ (ADD_FILL || CC_Hash)                         /* 0x0000017D */ \
+ (ADD_FILL || CC_PCR_Read)                     /* 0x0000017E */ \
+ (ADD_FILL || CC_PolicyPCR)                    /* 0x0000017F */ \
+ (ADD_FILL || CC_PolicyRestart)                /* 0x00000180 */ \
+ (ADD_FILL || CC_ReadClock)                    /* 0x00000181 */ \
+ (ADD_FILL || CC_PCR_Extend)                   /* 0x00000182 */ \
+ (ADD_FILL || CC_PCR_SetAuthValue)             /* 0x00000183 */ \
+ (ADD_FILL || CC_NV_Certify)                   /* 0x00000184 */ \
+ (ADD_FILL || CC_EventSequenceComplete)        /* 0x00000185 */ \
+ (ADD_FILL || CC_HashSequenceStart)            /* 0x00000186 */ \
+ (ADD_FILL || CC_PolicyPhysicalPresence)       /* 0x00000187 */ \
+ (ADD_FILL || CC_PolicyDuplicationSelect)      /* 0x00000188 */ \
+ (ADD_FILL || CC_PolicyGetDigest)              /* 0x00000189 */ \
+ (ADD_FILL || CC_TestParms)                    /* 0x0000018A */ \
+ (ADD_FILL || CC_Commit)                       /* 0x0000018B */ \
+ (ADD_FILL || CC_PolicyPassword)               /* 0x0000018C */ \
+ (ADD_FILL || CC_ZGen_2Phase)                  /* 0x0000018D */ \
+ (ADD_FILL || CC_EC_Ephemeral)                 /* 0x0000018E */ \
+ (ADD_FILL || CC_PolicyNvWritten)              /* 0x0000018F */ \
+ (ADD_FILL || CC_PolicyTemplate)               /* 0x00000190 */ \
+ (ADD_FILL || CC_CreateLoaded)                 /* 0x00000191 */ \
+ (ADD_FILL || CC_PolicyAuthorizeNV)            /* 0x00000192 */ \
```

```
     + (ADD_FILL || CC_EncryptDecrypt2)              /* 0x00000193 */ \
     + (ADD_FILL || CC_AC_GetCapability)             /* 0x00000194 */ \
     + (ADD_FILL || CC_AC_Send)                      /* 0x00000195 */ \
     + (ADD_FILL || CC_Policy_AC_SendSelect)         /* 0x00000196 */ \
     + (ADD_FILL || CC_CertifyX509)                  /* 0x00000197 */ \
     + (ADD_FILL || CC_ACT_SetTimeout)               /* 0x00000198 */ \
     + (ADD_FILL || CC_ECC_Encrypt)                  /* 0x00000199 */ \
     + (ADD_FILL || CC_ECC_Decrypt)                  /* 0x0000019A */ \
     + (ADD_FILL || CC_PolicyCapability)             /* 0x0000019B */ \
     + (ADD_FILL || CC_PolicyParameters)             /* 0x0000019C */ \
     + (ADD_FILL || CC_NV_DefineSpace2)              /* 0x0000019D */ \
     + (ADD_FILL || CC_NV_ReadPublic2)               /* 0x0000019E */ \
     + (ADD_FILL || CC_SetCapability)                /* 0x0000019F */ \
    )
#define VENDOR_COMMAND_ARRAY_SIZE (CC_Vendor_TCG_Test)
#define COMMAND_COUNT             (LIBRARY_COMMAND_ARRAY_SIZE +
VENDOR_COMMAND_ARRAY_SIZE)

// Table "Definition of TPM_RC Constants" (Part 2: Structures)
typedef UINT32 TPM_RC;
#define TYPE_OF_TPM_RC           UINT32
#define TPM_RC_SUCCESS           (TPM_RC)(0x000)
#define TPM_RC_BAD_TAG           (TPM_RC)(0x01E)
#define RC_VER1                  (TPM_RC)(0x100)
#define TPM_RC_INITIALIZE        (TPM_RC)(RC_VER1 + 0x000)
#define TPM_RC_FAILURE           (TPM_RC)(RC_VER1 + 0x001)
#define TPM_RC_SEQUENCE          (TPM_RC)(RC_VER1 + 0x003)
#define TPM_RC_PRIVATE           (TPM_RC)(RC_VER1 + 0x00B)
#define TPM_RC_HMAC              (TPM_RC)(RC_VER1 + 0x019)
#define TPM_RC_DISABLED          (TPM_RC)(RC_VER1 + 0x020)
#define TPM_RC_EXCLUSIVE         (TPM_RC)(RC_VER1 + 0x021)
#define TPM_RC_AUTH_TYPE         (TPM_RC)(RC_VER1 + 0x024)
#define TPM_RC_AUTH_MISSING      (TPM_RC)(RC_VER1 + 0x025)
#define TPM_RC_POLICY            (TPM_RC)(RC_VER1 + 0x026)
#define TPM_RC_PCR               (TPM_RC)(RC_VER1 + 0x027)
#define TPM_RC_PCR_CHANGED       (TPM_RC)(RC_VER1 + 0x028)
#define TPM_RC_UPGRADE           (TPM_RC)(RC_VER1 + 0x02D)
#define TPM_RC_TOO_MANY_CONTEXTS (TPM_RC)(RC_VER1 + 0x02E)
#define TPM_RC_AUTH_UNAVAILABLE  (TPM_RC)(RC_VER1 + 0x02F)
#define TPM_RC_REBOOT            (TPM_RC)(RC_VER1 + 0x030)
#define TPM_RC_UNBALANCED        (TPM_RC)(RC_VER1 + 0x031)
#define TPM_RC_COMMAND_SIZE      (TPM_RC)(RC_VER1 + 0x042)
#define TPM_RC_COMMAND_CODE      (TPM_RC)(RC_VER1 + 0x043)
#define TPM_RC_AUTHSIZE          (TPM_RC)(RC_VER1 + 0x044)
#define TPM_RC_AUTH_CONTEXT      (TPM_RC)(RC_VER1 + 0x045)
#define TPM_RC_NV_RANGE          (TPM_RC)(RC_VER1 + 0x046)
#define TPM_RC_NV_SIZE           (TPM_RC)(RC_VER1 + 0x047)
#define TPM_RC_NV_LOCKED         (TPM_RC)(RC_VER1 + 0x048)
#define TPM_RC_NV_AUTHORIZATION  (TPM_RC)(RC_VER1 + 0x049)
#define TPM_RC_NV_UNINITIALIZED  (TPM_RC)(RC_VER1 + 0x04A)
#define TPM_RC_NV_SPACE          (TPM_RC)(RC_VER1 + 0x04B)
#define TPM_RC_NV_DEFINED        (TPM_RC)(RC_VER1 + 0x04C)
#define TPM_RC_BAD_CONTEXT       (TPM_RC)(RC_VER1 + 0x050)
#define TPM_RC_CPHASH            (TPM_RC)(RC_VER1 + 0x051)
#define TPM_RC_PARENT            (TPM_RC)(RC_VER1 + 0x052)
#define TPM_RC_NEEDS_TEST        (TPM_RC)(RC_VER1 + 0x053)
#define TPM_RC_NO_RESULT         (TPM_RC)(RC_VER1 + 0x054)
#define TPM_RC_SENSITIVE         (TPM_RC)(RC_VER1 + 0x055)
#define RC_MAX_FM0               (TPM_RC)(RC_VER1 + 0x07F)
#define RC_FMT1                  (TPM_RC)(0x080)
#define TPM_RC_ASYMMETRIC        (TPM_RC)(RC_FMT1 + 0x001)
#define TPM_RCS_ASYMMETRIC       (TPM_RC)(RC_FMT1 + 0x001)
#define TPM_RC_ATTRIBUTES        (TPM_RC)(RC_FMT1 + 0x002)
#define TPM_RCS_ATTRIBUTES       (TPM_RC)(RC_FMT1 + 0x002)
#define TPM_RC_HASH              (TPM_RC)(RC_FMT1 + 0x003)
#define TPM_RCS_HASH             (TPM_RC)(RC_FMT1 + 0x003)
```

```
#define TPM_RC_VALUE              (TPM_RC)(RC_FMT1 + 0x004)
#define TPM_RCS_VALUE             (TPM_RC)(RC_FMT1 + 0x004)
#define TPM_RC_HIERARCHY          (TPM_RC)(RC_FMT1 + 0x005)
#define TPM_RCS_HIERARCHY         (TPM_RC)(RC_FMT1 + 0x005)
#define TPM_RC_KEY_SIZE           (TPM_RC)(RC_FMT1 + 0x007)
#define TPM_RCS_KEY_SIZE          (TPM_RC)(RC_FMT1 + 0x007)
#define TPM_RC_MGF                (TPM_RC)(RC_FMT1 + 0x008)
#define TPM_RCS_MGF               (TPM_RC)(RC_FMT1 + 0x008)
#define TPM_RC_MODE               (TPM_RC)(RC_FMT1 + 0x009)
#define TPM_RCS_MODE              (TPM_RC)(RC_FMT1 + 0x009)
#define TPM_RC_TYPE               (TPM_RC)(RC_FMT1 + 0x00A)
#define TPM_RCS_TYPE              (TPM_RC)(RC_FMT1 + 0x00A)
#define TPM_RC_HANDLE             (TPM_RC)(RC_FMT1 + 0x00B)
#define TPM_RCS_HANDLE            (TPM_RC)(RC_FMT1 + 0x00B)
#define TPM_RC_KDF                (TPM_RC)(RC_FMT1 + 0x00C)
#define TPM_RCS_KDF               (TPM_RC)(RC_FMT1 + 0x00C)
#define TPM_RC_RANGE              (TPM_RC)(RC_FMT1 + 0x00D)
#define TPM_RCS_RANGE             (TPM_RC)(RC_FMT1 + 0x00D)
#define TPM_RC_AUTH_FAIL          (TPM_RC)(RC_FMT1 + 0x00E)
#define TPM_RCS_AUTH_FAIL         (TPM_RC)(RC_FMT1 + 0x00E)
#define TPM_RC_NONCE              (TPM_RC)(RC_FMT1 + 0x00F)
#define TPM_RCS_NONCE             (TPM_RC)(RC_FMT1 + 0x00F)
#define TPM_RC_PP                 (TPM_RC)(RC_FMT1 + 0x010)
#define TPM_RCS_PP                (TPM_RC)(RC_FMT1 + 0x010)
#define TPM_RC_SCHEME             (TPM_RC)(RC_FMT1 + 0x012)
#define TPM_RCS_SCHEME            (TPM_RC)(RC_FMT1 + 0x012)
#define TPM_RC_SIZE               (TPM_RC)(RC_FMT1 + 0x015)
#define TPM_RCS_SIZE              (TPM_RC)(RC_FMT1 + 0x015)
#define TPM_RC_SYMMETRIC          (TPM_RC)(RC_FMT1 + 0x016)
#define TPM_RCS_SYMMETRIC         (TPM_RC)(RC_FMT1 + 0x016)
#define TPM_RC_TAG                (TPM_RC)(RC_FMT1 + 0x017)
#define TPM_RCS_TAG               (TPM_RC)(RC_FMT1 + 0x017)
#define TPM_RC_SELECTOR           (TPM_RC)(RC_FMT1 + 0x018)
#define TPM_RCS_SELECTOR          (TPM_RC)(RC_FMT1 + 0x018)
#define TPM_RC_INSUFFICIENT       (TPM_RC)(RC_FMT1 + 0x01A)
#define TPM_RCS_INSUFFICIENT      (TPM_RC)(RC_FMT1 + 0x01A)
#define TPM_RC_SIGNATURE          (TPM_RC)(RC_FMT1 + 0x01B)
#define TPM_RCS_SIGNATURE         (TPM_RC)(RC_FMT1 + 0x01B)
#define TPM_RC_KEY                (TPM_RC)(RC_FMT1 + 0x01C)
#define TPM_RCS_KEY               (TPM_RC)(RC_FMT1 + 0x01C)
#define TPM_RC_POLICY_FAIL        (TPM_RC)(RC_FMT1 + 0x01D)
#define TPM_RCS_POLICY_FAIL       (TPM_RC)(RC_FMT1 + 0x01D)
#define TPM_RC_INTEGRITY          (TPM_RC)(RC_FMT1 + 0x01F)
#define TPM_RCS_INTEGRITY         (TPM_RC)(RC_FMT1 + 0x01F)
#define TPM_RC_TICKET             (TPM_RC)(RC_FMT1 + 0x020)
#define TPM_RCS_TICKET            (TPM_RC)(RC_FMT1 + 0x020)
#define TPM_RC_RESERVED_BITS      (TPM_RC)(RC_FMT1 + 0x021)
#define TPM_RCS_RESERVED_BITS     (TPM_RC)(RC_FMT1 + 0x021)
#define TPM_RC_BAD_AUTH           (TPM_RC)(RC_FMT1 + 0x022)
#define TPM_RCS_BAD_AUTH          (TPM_RC)(RC_FMT1 + 0x022)
#define TPM_RC_EXPIRED            (TPM_RC)(RC_FMT1 + 0x023)
#define TPM_RCS_EXPIRED           (TPM_RC)(RC_FMT1 + 0x023)
#define TPM_RC_POLICY_CC          (TPM_RC)(RC_FMT1 + 0x024)
#define TPM_RCS_POLICY_CC         (TPM_RC)(RC_FMT1 + 0x024)
#define TPM_RC_BINDING            (TPM_RC)(RC_FMT1 + 0x025)
#define TPM_RCS_BINDING           (TPM_RC)(RC_FMT1 + 0x025)
#define TPM_RC_CURVE              (TPM_RC)(RC_FMT1 + 0x026)
#define TPM_RCS_CURVE             (TPM_RC)(RC_FMT1 + 0x026)
#define TPM_RC_ECC_POINT          (TPM_RC)(RC_FMT1 + 0x027)
#define TPM_RCS_ECC_POINT         (TPM_RC)(RC_FMT1 + 0x027)
#define TPM_RC_FW_LIMITED         (TPM_RC)(RC_FMT1 + 0x028)
#define TPM_RC_SVN_LIMITED        (TPM_RC)(RC_FMT1 + 0x029)
#define RC_WARN                   (TPM_RC)(0x900)
#define TPM_RC_CONTEXT_GAP        (TPM_RC)(RC_WARN + 0x001)
#define TPM_RC_OBJECT_MEMORY      (TPM_RC)(RC_WARN + 0x002)
#define TPM_RC_SESSION_MEMORY     (TPM_RC)(RC_WARN + 0x003)
```

```
#define TPM_RC_MEMORY             (TPM_RC)(RC_WARN + 0x004)
#define TPM_RC_SESSION_HANDLES    (TPM_RC)(RC_WARN + 0x005)
#define TPM_RC_OBJECT_HANDLES     (TPM_RC)(RC_WARN + 0x006)
#define TPM_RC_LOCALITY           (TPM_RC)(RC_WARN + 0x007)
#define TPM_RC_YIELDED            (TPM_RC)(RC_WARN + 0x008)
#define TPM_RC_CANCELED           (TPM_RC)(RC_WARN + 0x009)
#define TPM_RC_TESTING            (TPM_RC)(RC_WARN + 0x00A)
#define TPM_RC_REFERENCE_H0       (TPM_RC)(RC_WARN + 0x010)
#define TPM_RC_REFERENCE_H1       (TPM_RC)(RC_WARN + 0x011)
#define TPM_RC_REFERENCE_H2       (TPM_RC)(RC_WARN + 0x012)
#define TPM_RC_REFERENCE_H3       (TPM_RC)(RC_WARN + 0x013)
#define TPM_RC_REFERENCE_H4       (TPM_RC)(RC_WARN + 0x014)
#define TPM_RC_REFERENCE_H5       (TPM_RC)(RC_WARN + 0x015)
#define TPM_RC_REFERENCE_H6       (TPM_RC)(RC_WARN + 0x016)
#define TPM_RC_REFERENCE_S0       (TPM_RC)(RC_WARN + 0x018)
#define TPM_RC_REFERENCE_S1       (TPM_RC)(RC_WARN + 0x019)
#define TPM_RC_REFERENCE_S2       (TPM_RC)(RC_WARN + 0x01A)
#define TPM_RC_REFERENCE_S3       (TPM_RC)(RC_WARN + 0x01B)
#define TPM_RC_REFERENCE_S4       (TPM_RC)(RC_WARN + 0x01C)
#define TPM_RC_REFERENCE_S5       (TPM_RC)(RC_WARN + 0x01D)
#define TPM_RC_REFERENCE_S6       (TPM_RC)(RC_WARN + 0x01E)
#define TPM_RC_NV_RATE            (TPM_RC)(RC_WARN + 0x020)
#define TPM_RC_LOCKOUT            (TPM_RC)(RC_WARN + 0x021)
#define TPM_RC_RETRY              (TPM_RC)(RC_WARN + 0x022)
#define TPM_RC_NV_UNAVAILABLE     (TPM_RC)(RC_WARN + 0x023)
#define TPM_RC_NOT_USED           (TPM_RC)(RC_WARN + 0x7F)
#define TPM_RC_H                  (TPM_RC)(0x000)
#define TPM_RC_P                  (TPM_RC)(0x040)
#define TPM_RC_S                  (TPM_RC)(0x800)
#define TPM_RC_1                  (TPM_RC)(0x100)
#define TPM_RC_2                  (TPM_RC)(0x200)
#define TPM_RC_3                  (TPM_RC)(0x300)
#define TPM_RC_4                  (TPM_RC)(0x400)
#define TPM_RC_5                  (TPM_RC)(0x500)
#define TPM_RC_6                  (TPM_RC)(0x600)
#define TPM_RC_7                  (TPM_RC)(0x700)
#define TPM_RC_8                  (TPM_RC)(0x800)
#define TPM_RC_9                  (TPM_RC)(0x900)
#define TPM_RC_A                  (TPM_RC)(0xA00)
#define TPM_RC_B                  (TPM_RC)(0xB00)
#define TPM_RC_C                  (TPM_RC)(0xC00)
#define TPM_RC_D                  (TPM_RC)(0xD00)
#define TPM_RC_E                  (TPM_RC)(0xE00)
#define TPM_RC_F                  (TPM_RC)(0xF00)
#define TPM_RC_N_MASK             (TPM_RC)(0xF00)

// Table "Definition of TPM_CLOCK_ADJUST Constants" (Part 2: Structures)
typedef INT8 TPM_CLOCK_ADJUST;
#define TYPE_OF_TPM_CLOCK_ADJUST INT8
#define TPM_CLOCK_COARSE_SLOWER  (TPM_CLOCK_ADJUST)(-3)
#define TPM_CLOCK_MEDIUM_SLOWER  (TPM_CLOCK_ADJUST)(-2)
#define TPM_CLOCK_FINE_SLOWER    (TPM_CLOCK_ADJUST)(-1)
#define TPM_CLOCK_NO_CHANGE      (TPM_CLOCK_ADJUST)(0)
#define TPM_CLOCK_FINE_FASTER    (TPM_CLOCK_ADJUST)(1)
#define TPM_CLOCK_MEDIUM_FASTER  (TPM_CLOCK_ADJUST)(2)
#define TPM_CLOCK_COARSE_FASTER  (TPM_CLOCK_ADJUST)(3)

// Table "Definition of TPM_EO Constants" (Part 2: Structures)
typedef UINT16 TPM_EO;
#define TYPE_OF_TPM_EO     UINT16
#define TPM_EO_EQ          (TPM_EO)(0x0000)
#define TPM_EO_NEQ         (TPM_EO)(0x0001)
#define TPM_EO_SIGNED_GT   (TPM_EO)(0x0002)
#define TPM_EO_UNSIGNED_GT (TPM_EO)(0x0003)
#define TPM_EO_SIGNED_LT   (TPM_EO)(0x0004)
#define TPM_EO_UNSIGNED_LT (TPM_EO)(0x0005)
```

```
#define TPM_EO_SIGNED_GE    (TPM_EO)(0x0006)
#define TPM_EO_UNSIGNED_GE  (TPM_EO)(0x0007)
#define TPM_EO_SIGNED_LE    (TPM_EO)(0x0008)
#define TPM_EO_UNSIGNED_LE  (TPM_EO)(0x0009)
#define TPM_EO_BITSET       (TPM_EO)(0x000A)
#define TPM_EO_BITCLEAR     (TPM_EO)(0x000B)

// Table "Definition of TPM_ST Constants" (Part 2: Structures)
typedef UINT16 TPM_ST;
#define TYPE_OF_TPM_ST               UINT16
#define TPM_ST_RSP_COMMAND           (TPM_ST)(0x00C4)
#define TPM_ST_NULL                  (TPM_ST)(0x8000)
#define TPM_ST_NO_SESSIONS           (TPM_ST)(0x8001)
#define TPM_ST_SESSIONS              (TPM_ST)(0x8002)
#define TPM_ST_ATTEST_NV             (TPM_ST)(0x8014)
#define TPM_ST_ATTEST_COMMAND_AUDIT  (TPM_ST)(0x8015)
#define TPM_ST_ATTEST_SESSION_AUDIT  (TPM_ST)(0x8016)
#define TPM_ST_ATTEST_CERTIFY        (TPM_ST)(0x8017)
#define TPM_ST_ATTEST_QUOTE          (TPM_ST)(0x8018)
#define TPM_ST_ATTEST_TIME           (TPM_ST)(0x8019)
#define TPM_ST_ATTEST_CREATION       (TPM_ST)(0x801A)
#define TPM_ST_ATTEST_NV_DIGEST      (TPM_ST)(0x801C)
#define TPM_ST_CREATION              (TPM_ST)(0x8021)
#define TPM_ST_VERIFIED              (TPM_ST)(0x8022)
#define TPM_ST_AUTH_SECRET           (TPM_ST)(0x8023)
#define TPM_ST_HASHCHECK             (TPM_ST)(0x8024)
#define TPM_ST_AUTH_SIGNED           (TPM_ST)(0x8025)
#define TPM_ST_FU_MANIFEST           (TPM_ST)(0x8029)

// Table "Definition of TPM_SU Constants" (Part 2: Structures)
typedef UINT16 TPM_SU;
#define TYPE_OF_TPM_SU UINT16
#define TPM_SU_CLEAR    (TPM_SU)(0x0000)
#define TPM_SU_STATE    (TPM_SU)(0x0001)

// Table "Definition of TPM_SE Constants" (Part 2: Structures)
typedef UINT8 TPM_SE;
#define TYPE_OF_TPM_SE UINT8
#define TPM_SE_HMAC     (TPM_SE)(0x00)
#define TPM_SE_POLICY   (TPM_SE)(0x01)
#define TPM_SE_TRIAL    (TPM_SE)(0x03)

// Table "Definition of TPM_CAP Constants" (Part 2: Structures)
typedef UINT32 TPM_CAP;
#define TYPE_OF_TPM_CAP          UINT32
#define TPM_CAP_FIRST            (TPM_CAP)(0x00000000)
#define TPM_CAP_ALGS             (TPM_CAP)(0x00000000)
#define TPM_CAP_HANDLES          (TPM_CAP)(0x00000001)
#define TPM_CAP_COMMANDS         (TPM_CAP)(0x00000002)
#define TPM_CAP_PP_COMMANDS      (TPM_CAP)(0x00000003)
#define TPM_CAP_AUDIT_COMMANDS   (TPM_CAP)(0x00000004)
#define TPM_CAP_PCRS             (TPM_CAP)(0x00000005)
#define TPM_CAP_TPM_PROPERTIES   (TPM_CAP)(0x00000006)
#define TPM_CAP_PCR_PROPERTIES   (TPM_CAP)(0x00000007)
#define TPM_CAP_ECC_CURVES       (TPM_CAP)(0x00000008)
#define TPM_CAP_AUTH_POLICIES    (TPM_CAP)(0x00000009)
#define TPM_CAP_ACT              (TPM_CAP)(0x0000000A)
#define TPM_CAP_LAST             (TPM_CAP)(0x0000000A)
#define TPM_CAP_VENDOR_PROPERTY  (TPM_CAP)(0x00000100)

// Table "Definition of TPM_PT Constants" (Part 2: Structures)
typedef UINT32 TPM_PT;
#define TYPE_OF_TPM_PT             UINT32
#define TPM_PT_NONE                (TPM_PT)(0x00000000)
#define PT_GROUP                   (TPM_PT)(0x00000100)
#define PT_FIXED                   (TPM_PT)(PT_GROUP * 1)
```

```
#define TPM_PT_FAMILY_INDICATOR    (TPM_PT)(PT_FIXED + 0)
#define TPM_PT_LEVEL               (TPM_PT)(PT_FIXED + 1)
#define TPM_PT_REVISION            (TPM_PT)(PT_FIXED + 2)
#define TPM_PT_DAY_OF_YEAR         (TPM_PT)(PT_FIXED + 3)
#define TPM_PT_YEAR                (TPM_PT)(PT_FIXED + 4)
#define TPM_PT_MANUFACTURER        (TPM_PT)(PT_FIXED + 5)
#define TPM_PT_VENDOR_STRING_1     (TPM_PT)(PT_FIXED + 6)
#define TPM_PT_VENDOR_STRING_2     (TPM_PT)(PT_FIXED + 7)
#define TPM_PT_VENDOR_STRING_3     (TPM_PT)(PT_FIXED + 8)
#define TPM_PT_VENDOR_STRING_4     (TPM_PT)(PT_FIXED + 9)
#define TPM_PT_VENDOR_TPM_TYPE     (TPM_PT)(PT_FIXED + 10)
#define TPM_PT_FIRMWARE_VERSION_1  (TPM_PT)(PT_FIXED + 11)
#define TPM_PT_FIRMWARE_VERSION_2  (TPM_PT)(PT_FIXED + 12)
#define TPM_PT_INPUT_BUFFER        (TPM_PT)(PT_FIXED + 13)
#define TPM_PT_HR_TRANSIENT_MIN    (TPM_PT)(PT_FIXED + 14)
#define TPM_PT_HR_PERSISTENT_MIN   (TPM_PT)(PT_FIXED + 15)
#define TPM_PT_HR_LOADED_MIN       (TPM_PT)(PT_FIXED + 16)
#define TPM_PT_ACTIVE_SESSIONS_MAX (TPM_PT)(PT_FIXED + 17)
#define TPM_PT_PCR_COUNT           (TPM_PT)(PT_FIXED + 18)
#define TPM_PT_PCR_SELECT_MIN      (TPM_PT)(PT_FIXED + 19)
#define TPM_PT_CONTEXT_GAP_MAX     (TPM_PT)(PT_FIXED + 20)
#define TPM_PT_NV_COUNTERS_MAX     (TPM_PT)(PT_FIXED + 22)
#define TPM_PT_NV_INDEX_MAX        (TPM_PT)(PT_FIXED + 23)
#define TPM_PT_MEMORY              (TPM_PT)(PT_FIXED + 24)
#define TPM_PT_CLOCK_UPDATE        (TPM_PT)(PT_FIXED + 25)
#define TPM_PT_CONTEXT_HASH        (TPM_PT)(PT_FIXED + 26)
#define TPM_PT_CONTEXT_SYM         (TPM_PT)(PT_FIXED + 27)
#define TPM_PT_CONTEXT_SYM_SIZE    (TPM_PT)(PT_FIXED + 28)
#define TPM_PT_ORDERLY_COUNT       (TPM_PT)(PT_FIXED + 29)
#define TPM_PT_MAX_COMMAND_SIZE    (TPM_PT)(PT_FIXED + 30)
#define TPM_PT_MAX_RESPONSE_SIZE   (TPM_PT)(PT_FIXED + 31)
#define TPM_PT_MAX_DIGEST          (TPM_PT)(PT_FIXED + 32)
#define TPM_PT_MAX_OBJECT_CONTEXT  (TPM_PT)(PT_FIXED + 33)
#define TPM_PT_MAX_SESSION_CONTEXT (TPM_PT)(PT_FIXED + 34)
#define TPM_PT_PS_FAMILY_INDICATOR (TPM_PT)(PT_FIXED + 35)
#define TPM_PT_PS_LEVEL            (TPM_PT)(PT_FIXED + 36)
#define TPM_PT_PS_REVISION         (TPM_PT)(PT_FIXED + 37)
#define TPM_PT_PS_DAY_OF_YEAR      (TPM_PT)(PT_FIXED + 38)
#define TPM_PT_PS_YEAR             (TPM_PT)(PT_FIXED + 39)
#define TPM_PT_SPLIT_MAX           (TPM_PT)(PT_FIXED + 40)
#define TPM_PT_TOTAL_COMMANDS      (TPM_PT)(PT_FIXED + 41)
#define TPM_PT_LIBRARY_COMMANDS    (TPM_PT)(PT_FIXED + 42)
#define TPM_PT_VENDOR_COMMANDS     (TPM_PT)(PT_FIXED + 43)
#define TPM_PT_NV_BUFFER_MAX       (TPM_PT)(PT_FIXED + 44)
#define TPM_PT_MODES               (TPM_PT)(PT_FIXED + 45)
#define TPM_PT_MAX_CAP_BUFFER      (TPM_PT)(PT_FIXED + 46)
#define TPM_PT_FIRMWARE_SVN        (TPM_PT)(PT_FIXED + 47)
#define TPM_PT_FIRMWARE_MAX_SVN    (TPM_PT)(PT_FIXED + 48)
#define PT_VAR                     (TPM_PT)(PT_GROUP * 2)
#define TPM_PT_PERMANENT           (TPM_PT)(PT_VAR + 0)
#define TPM_PT_STARTUP_CLEAR       (TPM_PT)(PT_VAR + 1)
#define TPM_PT_HR_NV_INDEX         (TPM_PT)(PT_VAR + 2)
#define TPM_PT_HR_LOADED           (TPM_PT)(PT_VAR + 3)
#define TPM_PT_HR_LOADED_AVAIL     (TPM_PT)(PT_VAR + 4)
#define TPM_PT_HR_ACTIVE           (TPM_PT)(PT_VAR + 5)
#define TPM_PT_HR_ACTIVE_AVAIL     (TPM_PT)(PT_VAR + 6)
#define TPM_PT_HR_TRANSIENT_AVAIL  (TPM_PT)(PT_VAR + 7)
#define TPM_PT_HR_PERSISTENT       (TPM_PT)(PT_VAR + 8)
#define TPM_PT_HR_PERSISTENT_AVAIL (TPM_PT)(PT_VAR + 9)
#define TPM_PT_NV_COUNTERS         (TPM_PT)(PT_VAR + 10)
#define TPM_PT_NV_COUNTERS_AVAIL   (TPM_PT)(PT_VAR + 11)
#define TPM_PT_ALGORITHM_SET       (TPM_PT)(PT_VAR + 12)
#define TPM_PT_LOADED_CURVES       (TPM_PT)(PT_VAR + 13)
#define TPM_PT_LOCKOUT_COUNTER     (TPM_PT)(PT_VAR + 14)
#define TPM_PT_MAX_AUTH_FAIL       (TPM_PT)(PT_VAR + 15)
#define TPM_PT_LOCKOUT_INTERVAL    (TPM_PT)(PT_VAR + 16)
```

```
#define TPM_PT_LOCKOUT_RECOVERY     (TPM_PT)(PT_VAR + 17)
#define TPM_PT_NV_WRITE_RECOVERY    (TPM_PT)(PT_VAR + 18)
#define TPM_PT_AUDIT_COUNTER_0      (TPM_PT)(PT_VAR + 19)
#define TPM_PT_AUDIT_COUNTER_1      (TPM_PT)(PT_VAR + 20)

// Table "Definition of TPM_PT_PCR Constants" (Part 2: Structures)
typedef UINT32 TPM_PT_PCR;
#define TYPE_OF_TPM_PT_PCR      UINT32
#define TPM_PT_PCR_FIRST        (TPM_PT_PCR)(0x00000000)
#define TPM_PT_PCR_SAVE         (TPM_PT_PCR)(0x00000000)
#define TPM_PT_PCR_EXTEND_L0    (TPM_PT_PCR)(0x00000001)
#define TPM_PT_PCR_RESET_L0     (TPM_PT_PCR)(0x00000002)
#define TPM_PT_PCR_EXTEND_L1    (TPM_PT_PCR)(0x00000003)
#define TPM_PT_PCR_RESET_L1     (TPM_PT_PCR)(0x00000004)
#define TPM_PT_PCR_EXTEND_L2    (TPM_PT_PCR)(0x00000005)
#define TPM_PT_PCR_RESET_L2     (TPM_PT_PCR)(0x00000006)
#define TPM_PT_PCR_EXTEND_L3    (TPM_PT_PCR)(0x00000007)
#define TPM_PT_PCR_RESET_L3     (TPM_PT_PCR)(0x00000008)
#define TPM_PT_PCR_EXTEND_L4    (TPM_PT_PCR)(0x00000009)
#define TPM_PT_PCR_RESET_L4     (TPM_PT_PCR)(0x0000000A)
#define TPM_PT_PCR_NO_INCREMENT (TPM_PT_PCR)(0x00000011)
#define TPM_PT_PCR_DRTM_RESET   (TPM_PT_PCR)(0x00000012)
#define TPM_PT_PCR_POLICY       (TPM_PT_PCR)(0x00000013)
#define TPM_PT_PCR_AUTH         (TPM_PT_PCR)(0x00000014)
#define TPM_PT_PCR_LAST         (TPM_PT_PCR)(0x00000014)

// Table "Definition of TPM_PS Constants" (Part 2: Structures)
typedef UINT32 TPM_PS;
#define TYPE_OF_TPM_PS        UINT32
#define TPM_PS_MAIN           (TPM_PS)(0x00000000)
#define TPM_PS_PC             (TPM_PS)(0x00000001)
#define TPM_PS_PDA            (TPM_PS)(0x00000002)
#define TPM_PS_CELL_PHONE     (TPM_PS)(0x00000003)
#define TPM_PS_SERVER         (TPM_PS)(0x00000004)
#define TPM_PS_PERIPHERAL     (TPM_PS)(0x00000005)
#define TPM_PS_TSS            (TPM_PS)(0x00000006)
#define TPM_PS_STORAGE        (TPM_PS)(0x00000007)
#define TPM_PS_AUTHENTICATION (TPM_PS)(0x00000008)
#define TPM_PS_EMBEDDED       (TPM_PS)(0x00000009)
#define TPM_PS_HARDCOPY       (TPM_PS)(0x0000000A)
#define TPM_PS_INFRASTRUCTURE (TPM_PS)(0x0000000B)
#define TPM_PS_VIRTUALIZATION (TPM_PS)(0x0000000C)
#define TPM_PS_TNC            (TPM_PS)(0x0000000D)
#define TPM_PS_MULTI_TENANT   (TPM_PS)(0x0000000E)
#define TPM_PS_TC             (TPM_PS)(0x0000000F)

// Table "Definition of Types for Handles" (Part 2: Structures)
typedef UINT32 TPM_HANDLE;
#define TYPE_OF_TPM_HANDLE UINT32

// Table "Definition of TPM_HT Constants" (Part 2: Structures)
typedef UINT8 TPM_HT;
#define TYPE_OF_TPM_HT        UINT8
#define TPM_HT_PCR            (TPM_HT)(0x00)
#define TPM_HT_NV_INDEX       (TPM_HT)(0x01)
#define TPM_HT_HMAC_SESSION   (TPM_HT)(0x02)
#define TPM_HT_LOADED_SESSION (TPM_HT)(0x02)
#define TPM_HT_POLICY_SESSION (TPM_HT)(0x03)
#define TPM_HT_SAVED_SESSION  (TPM_HT)(0x03)
#define TPM_HT_EXTERNAL_NV    (TPM_HT)(0x11)
#define TPM_HT_PERMANENT_NV   (TPM_HT)(0x12)
#define TPM_HT_PERMANENT      (TPM_HT)(0x40)
#define TPM_HT_TRANSIENT      (TPM_HT)(0x80)
#define TPM_HT_PERSISTENT     (TPM_HT)(0x81)
#define TPM_HT_AC             (TPM_HT)(0x90)
```

```
// Table "Definition of TPM_RH Constants" (Part 2: Structures)
typedef TPM_HANDLE TPM_RH;
#define TYPE_OF_TPM_RH              TPM_HANDLE
#define TPM_RH_FIRST                (TPM_RH)(0x40000000)
#define TPM_RH_SRK                  (TPM_RH)(0x40000000)
#define TPM_RH_OWNER                (TPM_RH)(0x40000001)
#define TPM_RH_REVOKE               (TPM_RH)(0x40000002)
#define TPM_RH_TRANSPORT            (TPM_RH)(0x40000003)
#define TPM_RH_OPERATOR             (TPM_RH)(0x40000004)
#define TPM_RH_ADMIN                (TPM_RH)(0x40000005)
#define TPM_RH_EK                   (TPM_RH)(0x40000006)
#define TPM_RH_NULL                 (TPM_RH)(0x40000007)
#define TPM_RH_UNASSIGNED           (TPM_RH)(0x40000008)
#define TPM_RS_PW                   (TPM_RH)(0x40000009)
#define TPM_RH_LOCKOUT              (TPM_RH)(0x4000000A)
#define TPM_RH_ENDORSEMENT          (TPM_RH)(0x4000000B)
#define TPM_RH_PLATFORM             (TPM_RH)(0x4000000C)
#define TPM_RH_PLATFORM_NV          (TPM_RH)(0x4000000D)
#define TPM_RH_AUTH_00              (TPM_RH)(0x40000010)
#define TPM_RH_AUTH_FF              (TPM_RH)(0x4000010F)
#define TPM_RH_ACT_0                (TPM_RH)(0x40000110)
#define TPM_RH_ACT_F                (TPM_RH)(0x4000011F)
#define TPM_RH_FW_OWNER             (TPM_RH)(0x40000140)
#define TPM_RH_FW_ENDORSEMENT       (TPM_RH)(0x40000141)
#define TPM_RH_FW_PLATFORM          (TPM_RH)(0x40000142)
#define TPM_RH_FW_NULL              (TPM_RH)(0x40000143)
#define TPM_RH_SVN_OWNER_BASE       (TPM_RH)(0x40010000)
#define TPM_RH_SVN_ENDORSEMENT_BASE (TPM_RH)(0x40020000)
#define TPM_RH_SVN_PLATFORM_BASE    (TPM_RH)(0x40030000)
#define TPM_RH_SVN_NULL_BASE        (TPM_RH)(0x40040000)
#define TPM_RH_LAST                 (TPM_RH)(0x4004FFFF)
// Note: 0x40010001-0x4001FFFF, 0x40020001-0x4002FFFF,
// 0x40030001-0x4003FFFF, and 0x40040001-0x4004FFFF are
// valid reserved handles, but are not returned from
// TPM2_GetCapability().

// Table "Definition of TPM_HC Constants" (Part 2: Structures)
typedef TPM_HANDLE TPM_HC;
#define TYPE_OF_TPM_HC        TPM_HANDLE
#define HR_HANDLE_MASK        (TPM_HC)(0x00FFFFFF)
#define HR_RANGE_MASK         (TPM_HC)(0xFF000000)
#define HR_SHIFT              (TPM_HC)(24)
#define HR_PCR                (TPM_HC)((TPM_HT_PCR << HR_SHIFT))
#define HR_HMAC_SESSION       (TPM_HC)((TPM_HT_HMAC_SESSION << HR_SHIFT))
#define HR_POLICY_SESSION     (TPM_HC)((TPM_HT_POLICY_SESSION << HR_SHIFT))
#define HR_TRANSIENT          (TPM_HC)((TPM_HT_TRANSIENT << HR_SHIFT))
#define HR_PERSISTENT         (TPM_HC)((TPM_HT_PERSISTENT << HR_SHIFT))
#define HR_NV_INDEX           (TPM_HC)((TPM_HT_NV_INDEX << HR_SHIFT))
#define HR_EXTERNAL_NV        (TPM_HC)((TPM_HT_EXTERNAL_NV << HR_SHIFT))
#define HR_PERMANENT_NV       (TPM_HC)((TPM_HT_PERMANENT_NV << HR_SHIFT))
#define HR_PERMANENT          (TPM_HC)((TPM_HT_PERMANENT << HR_SHIFT))
#define PCR_FIRST             (TPM_HC)((HR_PCR + 0))
#define PCR_LAST              (TPM_HC)((PCR_FIRST + IMPLEMENTATION_PCR - 1))
#define HMAC_SESSION_FIRST    (TPM_HC)((HR_HMAC_SESSION + 0))
#define HMAC_SESSION_LAST     (TPM_HC)((HMAC_SESSION_FIRST + MAX_ACTIVE_SESSIONS - 1))
#define LOADED_SESSION_FIRST  (TPM_HC)(HMAC_SESSION_FIRST)
#define LOADED_SESSION_LAST   (TPM_HC)(HMAC_SESSION_LAST)
#define POLICY_SESSION_FIRST  (TPM_HC)((HR_POLICY_SESSION + 0))
#define POLICY_SESSION_LAST   (TPM_HC)((POLICY_SESSION_FIRST + MAX_ACTIVE_SESSIONS -
1))
#define TRANSIENT_FIRST       (TPM_HC)((HR_TRANSIENT + 0))
#define ACTIVE_SESSION_FIRST  (TPM_HC)(POLICY_SESSION_FIRST)
#define ACTIVE_SESSION_LAST   (TPM_HC)(POLICY_SESSION_LAST)
#define TRANSIENT_LAST        (TPM_HC)((TRANSIENT_FIRST + MAX_LOADED_OBJECTS - 1))
#define PERSISTENT_FIRST      (TPM_HC)((HR_PERSISTENT + 0))
#define PERSISTENT_LAST       (TPM_HC)((PERSISTENT_FIRST + 0x00FFFFFF))
```

```
#define SVN_OWNER_FIRST       (TPM_HC)((TPM_RH_SVN_OWNER_BASE + 0x0000))
#define SVN_OWNER_LAST        (TPM_HC)((TPM_RH_SVN_OWNER_BASE + 0xFFFF))
#define SVN_ENDORSEMENT_FIRST (TPM_HC)((TPM_RH_SVN_ENDORSEMENT_BASE + 0x0000))
#define SVN_ENDORSEMENT_LAST  (TPM_HC)((TPM_RH_SVN_ENDORSEMENT_BASE + 0xFFFF))
#define SVN_PLATFORM_FIRST    (TPM_HC)((TPM_RH_SVN_PLATFORM_BASE + 0x0000))
#define SVN_PLATFORM_LAST     (TPM_HC)((TPM_RH_SVN_PLATFORM_BASE + 0xFFFF))
#define SVN_NULL_FIRST        (TPM_HC)((TPM_RH_SVN_NULL_BASE + 0x0000))
#define SVN_NULL_LAST         (TPM_HC)((TPM_RH_SVN_NULL_BASE + 0xFFFF))
#define PLATFORM_PERSISTENT   (TPM_HC)((PERSISTENT_FIRST + 0x00800000))
#define NV_INDEX_FIRST        (TPM_HC)((HR_NV_INDEX + 0))
#define NV_INDEX_LAST         (TPM_HC)((NV_INDEX_FIRST + 0x00FFFFFF))
#define EXTERNAL_NV_FIRST     (TPM_HC)((HR_EXTERNAL_NV + 0))
#define EXTERNAL_NV_LAST      (TPM_HC)((EXTERNAL_NV_FIRST + 0x00FFFFFF))
#define PERMANENT_NV_FIRST    (TPM_HC)((HR_PERMANENT_NV + 0))
#define PERMANENT_NV_LAST     (TPM_HC)((PERMANENT_NV_FIRST + 0x00FFFFFF))
#define PERMANENT_FIRST       (TPM_HC)(TPM_RH_FIRST)
#define PERMANENT_LAST        (TPM_HC)(TPM_RH_LAST)
#define HR_NV_AC              (TPM_HC)(((TPM_HT_NV_INDEX << HR_SHIFT) + 0xD00000))
#define NV_AC_FIRST           (TPM_HC)((HR_NV_AC + 0))
#define NV_AC_LAST            (TPM_HC)((HR_NV_AC + 0x0000FFFF))
#define HR_AC                 (TPM_HC)((TPM_HT_AC << HR_SHIFT))
#define AC_FIRST              (TPM_HC)((HR_AC + 0))
#define AC_LAST               (TPM_HC)((HR_AC + 0x0000FFFF))

// Table "Definition of TPMA_ALGORITHM Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_ALGORITHM      UINT32
#define TPMA_ALGORITHM_TO_UINT32(a) (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_ALGORITHM(a) (*((TPMA_ALGORITHM*)&(a)))
#define TPMA_ALGORITHM_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_ALGORITHM_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_ALGORITHM(i, a)      \
    {                                           \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a);     \
        i        = UINT32_TO_TPMA_ALGORITHM(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned asymmetric          : 1;
    unsigned symmetric           : 1;
    unsigned hash                : 1;
    unsigned object              : 1;
    unsigned Reserved_bits_at_4  : 4;
    unsigned signing             : 1;
    unsigned encrypting          : 1;
    unsigned method              : 1;
    unsigned Reserved_bits_at_11 : 21;
} TPMA_ALGORITHM;

// Initializer for the bit-field structure
#  define TPMA_ALGORITHM_INITIALIZER(asymmetric,                                \
                                     symmetric,                                 \
                                     hash,                                      \
                                     object,                                    \
                                     bits_at_4,                                 \
                                     signing,                                   \
                                     encrypting,                                \
                                     method,                                    \
                                     bits_at_11)                                \
      {                                                                         \
          asymmetric, symmetric, hash, object, bits_at_4, signing, encrypting, \
              method, bits_at_11                                                \
      }
#else  // USE_BIT_FIELD_STRUCTURES
```

```
// This implements Table "Definition of TPMA_ALGORITHM Bits" (Part 2: Structures)
using bit masking
typedef UINT32 TPMA_ALGORITHM;
#  define TPMA_ALGORITHM_asymmetric (TPMA_ALGORITHM)(1 << 0)
#  define TPMA_ALGORITHM_symmetric  (TPMA_ALGORITHM)(1 << 1)
#  define TPMA_ALGORITHM_hash       (TPMA_ALGORITHM)(1 << 2)
#  define TPMA_ALGORITHM_object     (TPMA_ALGORITHM)(1 << 3)
#  define TPMA_ALGORITHM_signing    (TPMA_ALGORITHM)(1 << 8)
#  define TPMA_ALGORITHM_encrypting (TPMA_ALGORITHM)(1 << 9)
#  define TPMA_ALGORITHM_method     (TPMA_ALGORITHM)(1 << 10)

//  This is the initializer for a TPMA_ALGORITHM bit array.
#  define TPMA_ALGORITHM_INITIALIZER(asymmetric,                          \
                                     symmetric,                           \
                                     hash,                                \
                                     object,                              \
                                     bits_at_4,                           \
                                     signing,                             \
                                     encrypting,                          \
                                     method,                              \
                                     bits_at_11)                          \
      (TPMA_ALGORITHM)((asymmetric << 0) + (symmetric << 1) + (hash << 2)   \
                       + (object << 3) + (signing << 8) + (encrypting << 9) \
                       + (method << 10))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_OBJECT Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_OBJECT      UINT32
#define TPMA_OBJECT_TO_UINT32(a) (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_OBJECT(a) (*((TPMA_OBJECT*)&(a)))
#define TPMA_OBJECT_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_OBJECT_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_OBJECT(i, a)      \
    {                                        \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a);  \
        i        = UINT32_TO_TPMA_OBJECT(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned Reserved_bit_at_0    : 1;
    unsigned fixedTPM             : 1;
    unsigned stClear              : 1;
    unsigned fixedFirmware        : 1;
    unsigned fixedParent          : 1;
    unsigned sensitiveDataOrigin  : 1;
    unsigned userWithAuth         : 1;
    unsigned adminWithPolicy      : 1;
    unsigned firmwareLimited      : 1;
    unsigned svnLimited           : 1;
    unsigned noDA                 : 1;
    unsigned encryptedDuplication : 1;
    unsigned Reserved_bits_at_12  : 4;
    unsigned restricted           : 1;
    unsigned decrypt              : 1;
    unsigned sign                 : 1;
    unsigned x509sign             : 1;
    unsigned Reserved_bits_at_20  : 12;
} TPMA_OBJECT;

#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_OBJECT Bits" (Part 2: Structures) using
bit masking
typedef UINT32 TPMA_OBJECT;
```

```
#  define TPMA_OBJECT_fixedTPM             (TPMA_OBJECT)(1 << 1)
#  define TPMA_OBJECT_stClear              (TPMA_OBJECT)(1 << 2)
#  define TPMA_OBJECT_fixedFirmware        (TPMA_OBJECT)(1 << 3)
#  define TPMA_OBJECT_fixedParent          (TPMA_OBJECT)(1 << 4)
#  define TPMA_OBJECT_sensitiveDataOrigin  (TPMA_OBJECT)(1 << 5)
#  define TPMA_OBJECT_userWithAuth         (TPMA_OBJECT)(1 << 6)
#  define TPMA_OBJECT_adminWithPolicy      (TPMA_OBJECT)(1 << 7)
#  define TPMA_OBJECT_firmwareLimited      (TPMA_OBJECT)(1 << 8)
#  define TPMA_OBJECT_svnLimited           (TPMA_OBJECT)(1 << 9)
#  define TPMA_OBJECT_noDA                 (TPMA_OBJECT)(1 << 10)
#  define TPMA_OBJECT_encryptedDuplication (TPMA_OBJECT)(1 << 11)
#  define TPMA_OBJECT_restricted           (TPMA_OBJECT)(1 << 16)
#  define TPMA_OBJECT_decrypt              (TPMA_OBJECT)(1 << 17)
#  define TPMA_OBJECT_sign                 (TPMA_OBJECT)(1 << 18)
#  define TPMA_OBJECT_x509sign             (TPMA_OBJECT)(1 << 19)

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_SESSION Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_SESSION     UINT8
#define TPMA_SESSION_TO_UINT8(a) (*((UINT8*)&(a)))
#define UINT8_TO_TPMA_SESSION(a) (*((TPMA_SESSION*)&(a)))
#define TPMA_SESSION_TO_BYTE_ARRAY(i, a) \
    UINT8_TO_BYTE_ARRAY((TPMA_SESSION_TO_UINT8(i)), (a))
#define BYTE_ARRAY_TO_TPMA_SESSION(i, a)    \
    {                                       \
        UINT8 x = BYTE_ARRAY_TO_UINT8(a);   \
        i       = UINT8_TO_TPMA_SESSION(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned continueSession    : 1;
    unsigned auditExclusive     : 1;
    unsigned auditReset         : 1;
    unsigned Reserved_bits_at_3 : 2;
    unsigned decrypt            : 1;
    unsigned encrypt            : 1;
    unsigned audit              : 1;
} TPMA_SESSION;

// Initializer for the bit-field structure
#  define TPMA_SESSION_INITIALIZER(continuesession,                                  \
                                   auditexclusive,                                   \
                                   auditreset,                                       \
                                   bits_at_3,                                        \
                                   decrypt,                                          \
                                   encrypt,                                          \
                                   audit)                                            \
      {                                                                              \
          continuesession, auditexclusive, auditreset, bits_at_3, decrypt, encrypt, \
              audit                                                                  \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_SESSION Bits" (Part 2: Structures) using
bit masking
typedef UINT8 TPMA_SESSION;
#  define TPMA_SESSION_continueSession (TPMA_SESSION)(1 << 0)
#  define TPMA_SESSION_auditExclusive  (TPMA_SESSION)(1 << 1)
#  define TPMA_SESSION_auditReset      (TPMA_SESSION)(1 << 2)
#  define TPMA_SESSION_decrypt         (TPMA_SESSION)(1 << 5)
#  define TPMA_SESSION_encrypt         (TPMA_SESSION)(1 << 6)
#  define TPMA_SESSION_audit           (TPMA_SESSION)(1 << 7)

//  This is the initializer for a TPMA_SESSION bit array.
```

```
#  define TPMA_SESSION_INITIALIZER(continuesession,                        \
                                    auditexclusive,                       \
                                    auditreset,                           \
                                    bits_at_3,                            \
                                    decrypt,                              \
                                    encrypt,                              \
                                    audit)                                \
      (TPMA_SESSION)((continuesession << 0) + (auditexclusive << 1)       \
                     + (auditreset << 2) + (decrypt << 5) + (encrypt << 6) \
                     + (audit << 7))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_LOCALITY Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_LOCALITY     UINT8
#define TPMA_LOCALITY_TO_UINT8(a) (*((UINT8*)&(a)))
#define UINT8_TO_TPMA_LOCALITY(a) (*((TPMA_LOCALITY*)&(a)))
#define TPMA_LOCALITY_TO_BYTE_ARRAY(i, a) \
    UINT8_TO_BYTE_ARRAY((TPMA_LOCALITY_TO_UINT8(i)), (a))
#define BYTE_ARRAY_TO_TPMA_LOCALITY(i, a)    \
    {                                        \
        UINT8 x = BYTE_ARRAY_TO_UINT8(a);    \
        i       = UINT8_TO_TPMA_LOCALITY(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned TPM_LOC_ZERO  : 1;
    unsigned TPM_LOC_ONE   : 1;
    unsigned TPM_LOC_TWO   : 1;
    unsigned TPM_LOC_THREE : 1;
    unsigned TPM_LOC_FOUR  : 1;
    unsigned Extended      : 3;
} TPMA_LOCALITY;

// Initializer for the bit-field structure
#  define TPMA_LOCALITY_INITIALIZER(                                                  \
      tpm_loc_zero, tpm_loc_one, tpm_loc_two, tpm_loc_three, tpm_loc_four, extended) \
      {                                                                               \
          tpm_loc_zero, tpm_loc_one, tpm_loc_two, tpm_loc_three, tpm_loc_four,       \
              extended                                                                \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_LOCALITY Bits" (Part 2: Structures) using
bit masking
typedef UINT8 TPMA_LOCALITY;
#  define TPMA_LOCALITY_TPM_LOC_ZERO    (TPMA_LOCALITY)(1 << 0)
#  define TPMA_LOCALITY_TPM_LOC_ONE     (TPMA_LOCALITY)(1 << 1)
#  define TPMA_LOCALITY_TPM_LOC_TWO     (TPMA_LOCALITY)(1 << 2)
#  define TPMA_LOCALITY_TPM_LOC_THREE   (TPMA_LOCALITY)(1 << 3)
#  define TPMA_LOCALITY_TPM_LOC_FOUR    (TPMA_LOCALITY)(1 << 4)
#  define TPMA_LOCALITY_Extended        (TPMA_LOCALITY)(7 << 5)
#  define TPMA_LOCALITY_Extended_SHIFT 5

//  This is the initializer for a TPMA_LOCALITY bit array.
#  define TPMA_LOCALITY_INITIALIZER(                                                  \
      tpm_loc_zero, tpm_loc_one, tpm_loc_two, tpm_loc_three, tpm_loc_four, extended) \
      (TPMA_LOCALITY)((tpm_loc_zero << 0) + (tpm_loc_one << 1) + (tpm_loc_two << 2)   \
                      + (tpm_loc_three << 3) + (tpm_loc_four << 4)                    \
                      + (extended << 5))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_PERMANENT Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_PERMANENT      UINT32
```

```
#define TPMA_PERMANENT_TO_UINT32(a) (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_PERMANENT(a) (*((TPMA_PERMANENT*)&(a)))
#define TPMA_PERMANENT_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_PERMANENT_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_PERMANENT(i, a)      \
    {                                           \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a);     \
        i        = UINT32_TO_TPMA_PERMANENT(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned ownerAuthSet        : 1;
    unsigned endorsementAuthSet  : 1;
    unsigned lockoutAuthSet      : 1;
    unsigned Reserved_bits_at_3  : 5;
    unsigned disableClear        : 1;
    unsigned inLockout           : 1;
    unsigned tpmGeneratedEPS     : 1;
    unsigned Reserved_bits_at_11 : 21;
} TPMA_PERMANENT;

// Initializer for the bit-field structure
#  define TPMA_PERMANENT_INITIALIZER(ownerauthset,                                  \
                                     endorsementauthset,                            \
                                     lockoutauthset,                                \
                                     bits_at_3,                                     \
                                     disableclear,                                  \
                                     inlockout,                                     \
                                     tpmgeneratedeps,                               \
                                     bits_at_11)                                    \
      {                                                                             \
          ownerauthset, endorsementauthset, lockoutauthset, bits_at_3, disableclear, \
              inlockout, tpmgeneratedeps, bits_at_11                                \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_PERMANENT Bits" (Part 2: Structures)
using bit masking
typedef UINT32 TPMA_PERMANENT;
#  define TPMA_PERMANENT_ownerAuthSet       (TPMA_PERMANENT)(1 << 0)
#  define TPMA_PERMANENT_endorsementAuthSet (TPMA_PERMANENT)(1 << 1)
#  define TPMA_PERMANENT_lockoutAuthSet     (TPMA_PERMANENT)(1 << 2)
#  define TPMA_PERMANENT_disableClear       (TPMA_PERMANENT)(1 << 8)
#  define TPMA_PERMANENT_inLockout          (TPMA_PERMANENT)(1 << 9)
#  define TPMA_PERMANENT_tpmGeneratedEPS    (TPMA_PERMANENT)(1 << 10)

//  This is the initializer for a TPMA_PERMANENT bit array.
#  define TPMA_PERMANENT_INITIALIZER(ownerauthset,                      \
                                     endorsementauthset,                \
                                     lockoutauthset,                    \
                                     bits_at_3,                         \
                                     disableclear,                      \
                                     inlockout,                         \
                                     tpmgeneratedeps,                   \
                                     bits_at_11)                        \
      (TPMA_PERMANENT)((ownerauthset << 0) + (endorsementauthset << 1) \
                       + (lockoutauthset << 2) + (disableclear << 8)   \
                       + (inlockout << 9) + (tpmgeneratedeps << 10))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_STARTUP_CLEAR Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_STARTUP_CLEAR      UINT32
#define TPMA_STARTUP_CLEAR_TO_UINT32(a) (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_STARTUP_CLEAR(a) (*((TPMA_STARTUP_CLEAR*)&(a)))
```

```
#define TPMA_STARTUP_CLEAR_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_STARTUP_CLEAR_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_STARTUP_CLEAR(i, a)      \
    {                                               \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a);         \
        i        = UINT32_TO_TPMA_STARTUP_CLEAR(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned phEnable           : 1;
    unsigned shEnable           : 1;
    unsigned ehEnable           : 1;
    unsigned phEnableNV         : 1;
    unsigned Reserved_bits_at_4 : 27;
    unsigned orderly            : 1;
} TPMA_STARTUP_CLEAR;

// Initializer for the bit-field structure
#  define TPMA_STARTUP_CLEAR_INITIALIZER(                                    \
      phenable, shenable, ehenable, phenablenv, bits_at_4, orderly)    \
      {                                                                \
          phenable, shenable, ehenable, phenablenv, bits_at_4, orderly \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_STARTUP_CLEAR Bits" (Part 2: Structures)
using bit masking
typedef UINT32 TPMA_STARTUP_CLEAR;
#  define TPMA_STARTUP_CLEAR_phEnable    (TPMA_STARTUP_CLEAR)(1 << 0)
#  define TPMA_STARTUP_CLEAR_shEnable    (TPMA_STARTUP_CLEAR)(1 << 1)
#  define TPMA_STARTUP_CLEAR_ehEnable    (TPMA_STARTUP_CLEAR)(1 << 2)
#  define TPMA_STARTUP_CLEAR_phEnableNV (TPMA_STARTUP_CLEAR)(1 << 3)
#  define TPMA_STARTUP_CLEAR_orderly     (TPMA_STARTUP_CLEAR)(1 << 31)

//  This is the initializer for a TPMA_STARTUP_CLEAR bit array.
#  define TPMA_STARTUP_CLEAR_INITIALIZER(                                      \
      phenable, shenable, ehenable, phenablenv, bits_at_4, orderly)            \
      (TPMA_STARTUP_CLEAR)((phenable << 0) + (shenable << 1) + (ehenable << 2) \
                           + (phenablenv << 3) + (orderly << 31))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_MEMORY Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_MEMORY      UINT32
#define TPMA_MEMORY_TO_UINT32(a)  (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_MEMORY(a)  (*((TPMA_MEMORY*)&(a)))
#define TPMA_MEMORY_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_MEMORY_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_MEMORY(i, a)      \
    {                                        \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a);  \
        i        = UINT32_TO_TPMA_MEMORY(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned sharedRAM          : 1;
    unsigned sharedNV           : 1;
    unsigned objectCopiedToRam  : 1;
    unsigned Reserved_bits_at_3 : 29;
} TPMA_MEMORY;

// Initializer for the bit-field structure
#  define TPMA_MEMORY_INITIALIZER(sharedram, sharednv, objectcopiedtoram, bits_at_3) \
      {                                                                              \
```

```
          sharedram, sharednv, objectcopiedtoram, bits_at_3                          \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_MEMORY Bits" (Part 2: Structures) using
bit masking
typedef UINT32 TPMA_MEMORY;
#  define TPMA_MEMORY_sharedRAM         (TPMA_MEMORY)(1 << 0)
#  define TPMA_MEMORY_sharedNV          (TPMA_MEMORY)(1 << 1)
#  define TPMA_MEMORY_objectCopiedToRam (TPMA_MEMORY)(1 << 2)

//  This is the initializer for a TPMA_MEMORY bit array.
#  define TPMA_MEMORY_INITIALIZER(sharedram, sharednv, objectcopiedtoram, bits_at_3) \
      (TPMA_MEMORY)((sharedram << 0) + (sharednv << 1) + (objectcopiedtoram << 2))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_CC Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_CC             UINT32
#define TPMA_CC_TO_UINT32(a)        (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_CC(a)        (*((TPMA_CC*)&(a)))
#define TPMA_CC_TO_BYTE_ARRAY(i, a) UINT32_TO_BYTE_ARRAY((TPMA_CC_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_CC(i, a)         \
    {                                       \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a); \
        i        = UINT32_TO_TPMA_CC(x);    \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned commandIndex        : 16;
    unsigned Reserved_bits_at_16 : 6;
    unsigned nv                  : 1;
    unsigned extensive           : 1;
    unsigned flushed             : 1;
    unsigned cHandles            : 3;
    unsigned rHandle             : 1;
    unsigned V                   : 1;
    unsigned Reserved_bits_at_30 : 2;
} TPMA_CC;

// Initializer for the bit-field structure
#  define TPMA_CC_INITIALIZER(commandindex,                                          \
                              bits_at_16,                                            \
                              nv,                                                    \
                              extensive,                                             \
                              flushed,                                               \
                              chandles,                                              \
                              rhandle,                                               \
                              v,                                                     \
                              bits_at_30)                                            \
      {                                                                              \
          commandindex, bits_at_16, nv, extensive, flushed, chandles, rhandle, v, \
              bits_at_30                                                             \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_CC Bits" (Part 2: Structures) using bit
masking
typedef TPM_CC TPMA_CC;
#  define TPMA_CC_commandIndex       (TPMA_CC)(0xFFFF << 0)
#  define TPMA_CC_commandIndex_SHIFT 0
#  define TPMA_CC_nv                 (TPMA_CC)(1 << 22)
#  define TPMA_CC_extensive          (TPMA_CC)(1 << 23)
#  define TPMA_CC_flushed            (TPMA_CC)(1 << 24)
#  define TPMA_CC_cHandles           (TPMA_CC)(7 << 25)
```

```
#  define TPMA_CC_cHandles_SHIFT     25
#  define TPMA_CC_rHandle            (TPMA_CC)(1 << 28)
#  define TPMA_CC_V                  (TPMA_CC)(1 << 29)

//  This is the initializer for a TPMA_CC bit array.
#  define TPMA_CC_INITIALIZER(commandindex,                          \
                              bits_at_16,                            \
                              nv,                                    \
                              extensive,                             \
                              flushed,                               \
                              chandles,                              \
                              rhandle,                               \
                              v,                                     \
                              bits_at_30)                            \
      (TPMA_CC)((commandindex << 0) + (nv << 22) + (extensive << 23) \
                + (flushed << 24) + (chandles << 25) + (rhandle << 28) + (v << 29))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_MODES Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_MODES      UINT32
#define TPMA_MODES_TO_UINT32(a) (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_MODES(a) (*((TPMA_MODES*)&(a)))
#define TPMA_MODES_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_MODES_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_MODES(i, a)      \
    {                                       \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a); \
        i        = UINT32_TO_TPMA_MODES(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned FIPS_140_2           : 1;
    unsigned FIPS_140_3           : 1;
    unsigned FIPS_140_3_INDICATOR : 2;
    unsigned Reserved_bits_at_4   : 28;
} TPMA_MODES;

// Initializer for the bit-field structure
#  define TPMA_MODES_INITIALIZER(                                  \
      fips_140_2, fips_140_3, fips_140_3_indicator, bits_at_4)     \
      {                                                            \
          fips_140_2, fips_140_3, fips_140_3_indicator, bits_at_4 \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_MODES Bits" (Part 2: Structures) using
bit masking
typedef UINT32 TPMA_MODES;
#  define TPMA_MODES_FIPS_140_2                (TPMA_MODES)(1 << 0)
#  define TPMA_MODES_FIPS_140_3                (TPMA_MODES)(1 << 1)
#  define TPMA_MODES_FIPS_140_3_INDICATOR      (TPMA_MODES)(3 << 2)
#  define TPMA_MODES_FIPS_140_3_INDICATOR_SHIFT 2

//  This is the initializer for a TPMA_MODES bit array.
#  define TPMA_MODES_INITIALIZER(                                \
      fips_140_2, fips_140_3, fips_140_3_indicator, bits_at_4) \
      (TPMA_MODES)(                                            \
          (fips_140_2 << 0) + (fips_140_3 << 1) + (fips_140_3_indicator << 2))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_X509_KEY_USAGE Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_X509_KEY_USAGE      UINT32
#define TPMA_X509_KEY_USAGE_TO_UINT32(a) (*((UINT32*)&(a)))
```

```
#define UINT32_TO_TPMA_X509_KEY_USAGE(a) (*((TPMA_X509_KEY_USAGE*)&(a)))
#define TPMA_X509_KEY_USAGE_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_X509_KEY_USAGE_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_X509_KEY_USAGE(i, a)      \
    {                                                \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a);          \
        i        = UINT32_TO_TPMA_X509_KEY_USAGE(x); \
    }
#define TPMA_X509_KEY_USAGE_ALLOWED_BITS (0xff800000)
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned Reserved_bits_at_0 : 23;
    unsigned decipherOnly       : 1;
    unsigned encipherOnly       : 1;
    unsigned cRLSign            : 1;
    unsigned keyCertSign        : 1;
    unsigned keyAgreement       : 1;
    unsigned dataEncipherment   : 1;
    unsigned keyEncipherment    : 1;
    unsigned nonrepudiation     : 1;
    unsigned digitalSignature   : 1;
} TPMA_X509_KEY_USAGE;

// Initializer for the bit-field structure
#  define TPMA_X509_KEY_USAGE_INITIALIZER(bits_at_0,                          \
                                          decipheronly,                       \
                                          encipheronly,                       \
                                          crlsign,                            \
                                          keycertsign,                        \
                                          keyagreement,                       \
                                          dataencipherment,                   \
                                          keyencipherment,                    \
                                          nonrepudiation,                     \
                                          digitalsignature)                   \
      {                                                                       \
          bits_at_0, decipheronly, encipheronly, crlsign, keycertsign, keyagreement, \
              dataencipherment, keyencipherment, nonrepudiation, digitalsignature    \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_X509_KEY_USAGE Bits" (Part 2: Structures)
using bit masking
typedef UINT32 TPMA_X509_KEY_USAGE;
#  define TPMA_X509_KEY_USAGE_decipherOnly     (TPMA_X509_KEY_USAGE)(1 << 23)
#  define TPMA_X509_KEY_USAGE_encipherOnly     (TPMA_X509_KEY_USAGE)(1 << 24)
#  define TPMA_X509_KEY_USAGE_cRLSign          (TPMA_X509_KEY_USAGE)(1 << 25)
#  define TPMA_X509_KEY_USAGE_keyCertSign      (TPMA_X509_KEY_USAGE)(1 << 26)
#  define TPMA_X509_KEY_USAGE_keyAgreement     (TPMA_X509_KEY_USAGE)(1 << 27)
#  define TPMA_X509_KEY_USAGE_dataEncipherment (TPMA_X509_KEY_USAGE)(1 << 28)
#  define TPMA_X509_KEY_USAGE_keyEncipherment  (TPMA_X509_KEY_USAGE)(1 << 29)
#  define TPMA_X509_KEY_USAGE_nonrepudiation   (TPMA_X509_KEY_USAGE)(1 << 30)
#  define TPMA_X509_KEY_USAGE_digitalSignature (TPMA_X509_KEY_USAGE)(1 << 31)

//  This is the initializer for a TPMA_X509_KEY_USAGE bit array.
#  define TPMA_X509_KEY_USAGE_INITIALIZER(bits_at_0,                        \
                                          decipheronly,                     \
                                          encipheronly,                     \
                                          crlsign,                          \
                                          keycertsign,                      \
                                          keyagreement,                     \
                                          dataencipherment,                 \
                                          keyencipherment,                  \
                                          nonrepudiation,                   \
                                          digitalsignature)                 \
      (TPMA_X509_KEY_USAGE)((decipheronly << 23) + (encipheronly << 24)     \
```

```
                            + (crlsign << 25) + (keycertsign << 26)            \
                            + (keyagreement << 27) + (dataencipherment << 28)  \
                            + (keyencipherment << 29) + (nonrepudiation << 30) \
                            + (digitalsignature << 31))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_ACT Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_ACT      UINT32
#define TPMA_ACT_TO_UINT32(a) (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_ACT(a) (*((TPMA_ACT*)&(a)))
#define TPMA_ACT_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPMA_ACT_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_ACT(i, a)        \
    {                                       \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a); \
        i        = UINT32_TO_TPMA_ACT(x);   \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned signaled           : 1;
    unsigned preserveSignaled   : 1;
    unsigned Reserved_bits_at_2 : 30;
} TPMA_ACT;

// Initializer for the bit-field structure
#  define TPMA_ACT_INITIALIZER(signaled, preservesignaled, bits_at_2) \
      {                                                               \
          signaled, preservesignaled, bits_at_2                       \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_ACT Bits" (Part 2: Structures) using bit
masking
typedef UINT32 TPMA_ACT;
#  define TPMA_ACT_signaled         (TPMA_ACT)(1 << 0)
#  define TPMA_ACT_preserveSignaled (TPMA_ACT)(1 << 1)

//  This is the initializer for a TPMA_ACT bit array.
#  define TPMA_ACT_INITIALIZER(signaled, preservesignaled, bits_at_2) \
      (TPMA_ACT)((signaled << 0) + (preservesignaled << 1))

#endif  // USE_BIT_FIELD_STRUCTURES

typedef BYTE       TPMI_YES_NO;               // (Part 2: Structures)
typedef TPM_HANDLE TPMI_DH_OBJECT;            // (Part 2: Structures)
typedef TPM_HANDLE TPMI_DH_PARENT;            // (Part 2: Structures)
typedef TPM_HANDLE TPMI_DH_PERSISTENT;        // (Part 2: Structures)
typedef TPM_HANDLE TPMI_DH_ENTITY;            // (Part 2: Structures)
typedef TPM_HANDLE TPMI_DH_PCR;               // (Part 2: Structures)
typedef TPM_HANDLE TPMI_SH_AUTH_SESSION;      // (Part 2: Structures)
typedef TPM_HANDLE TPMI_SH_HMAC;              // (Part 2: Structures)
typedef TPM_HANDLE TPMI_SH_POLICY;            // (Part 2: Structures)
typedef TPM_HANDLE TPMI_DH_CONTEXT;           // (Part 2: Structures)
typedef TPM_HANDLE TPMI_DH_SAVED;             // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_HIERARCHY;         // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_ENABLES;           // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_HIERARCHY_AUTH;    // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_HIERARCHY_POLICY;  // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_BASE_HIERARCHY;    // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_PLATFORM;          // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_OWNER;             // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_ENDORSEMENT;       // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_PROVISION;         // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_CLEAR;             // (Part 2: Structures)
```

```
typedef TPM_HANDLE TPMI_RH_NV_AUTH;            // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_LOCKOUT;            // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_NV_INDEX;           // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_NV_DEFINED_INDEX;   // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_NV_LEGACY_INDEX;    // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_NV_EXP_INDEX;       // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_AC;                 // (Part 2: Structures)
typedef TPM_HANDLE TPMI_RH_ACT;                // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_HASH;              // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_ASYM;              // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_SYM;               // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_SYM_OBJECT;        // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_SYM_MODE;          // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_KDF;               // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_SIG_SCHEME;        // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ECC_KEY_EXCHANGE;      // (Part 2: Structures)
typedef TPM_ST     TPMI_ST_COMMAND_TAG;        // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_MAC_SCHEME;        // (Part 2: Structures)
typedef TPM_ALG_ID TPMI_ALG_CIPHER_MODE;       // (Part 2: Structures)
typedef BYTE       TPMS_EMPTY;                 // (Part 2: Structures)

typedef struct
{  // (Part 2: Structures)
    TPM_ALG_ID     alg;
    TPMA_ALGORITHM attributes;
} TPMS_ALGORITHM_DESCRIPTION;

typedef union
{  // (Part 2: Structures)
#if ALG_SHA1
    BYTE sha1[SHA1_DIGEST_SIZE];
#endif  // ALG_SHA1
#if ALG_SHA256
    BYTE sha256[SHA256_DIGEST_SIZE];
#endif  // ALG_SHA256
#if ALG_SHA256_192
    BYTE sha256_192[SHA256_192_DIGEST_SIZE];
#endif  // ALG_SHA256_192
#if ALG_SHA3_256
    BYTE sha3_256[SHA3_256_DIGEST_SIZE];
#endif  // ALG_SHA3_256
#if ALG_SHA3_384
    BYTE sha3_384[SHA3_384_DIGEST_SIZE];
#endif  // ALG_SHA3_384
#if ALG_SHA3_512
    BYTE sha3_512[SHA3_512_DIGEST_SIZE];
#endif  // ALG_SHA3_512
#if ALG_SHA384
    BYTE sha384[SHA384_DIGEST_SIZE];
#endif  // ALG_SHA384
#if ALG_SHA512
    BYTE sha512[SHA512_DIGEST_SIZE];
#endif  // ALG_SHA512
#if ALG_SHAKE256_192
    BYTE shake256_192[SHAKE256_192_DIGEST_SIZE];
#endif  // ALG_SHAKE256_192
#if ALG_SHAKE256_256
    BYTE shake256_256[SHAKE256_256_DIGEST_SIZE];
#endif  // ALG_SHAKE256_256
#if ALG_SHAKE256_512
    BYTE shake256_512[SHAKE256_512_DIGEST_SIZE];
#endif  // ALG_SHAKE256_512
#if ALG_SM3_256
    BYTE sm3_256[SM3_256_DIGEST_SIZE];
#endif  // ALG_SM3_256
} TPMU_HA;
```

```
typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_HASH hashAlg;
    TPMU_HA       digest;
} TPMT_HA;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(TPMU_HA)];
    } t;
    TPM2B b;
} TPM2B_DIGEST;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(TPMT_HA)];
    } t;
    TPM2B b;
} TPM2B_DATA;

// Table "Definition of Types for TPM2B_NONCE" (Part 2: Structures)
typedef TPM2B_DIGEST TPM2B_NONCE;
#define TYPE_OF_TPM2B_NONCE TPM2B_DIGEST

// Table "Definition of Types for TPM2B_AUTH" (Part 2: Structures)
typedef TPM2B_DIGEST TPM2B_AUTH;
#define TYPE_OF_TPM2B_AUTH TPM2B_DIGEST

// Table "Definition of Types for TPM2B_OPERAND" (Part 2: Structures)
typedef TPM2B_DIGEST TPM2B_OPERAND;
#define TYPE_OF_TPM2B_OPERAND TPM2B_DIGEST

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[1024];
    } t;
    TPM2B b;
} TPM2B_EVENT;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[MAX_DIGEST_BUFFER];
    } t;
    TPM2B b;
} TPM2B_MAX_BUFFER;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[MAX_NV_BUFFER_SIZE];
    } t;
```

```
    TPM2B b;
} TPM2B_MAX_NV_BUFFER;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(UINT64)];
    } t;
    TPM2B b;
} TPM2B_TIMEOUT;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[MAX_SYM_BLOCK_SIZE];
    } t;
    TPM2B b;
} TPM2B_IV;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[512];
    } t;
    TPM2B b;
} TPM2B_VENDOR_PROPERTY;

typedef union
{  // (Part 2: Structures)
    TPMT_HA    digest;
    TPM_HANDLE handle;
} TPMU_NAME;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   name[sizeof(TPMU_NAME)];
    } t;
    TPM2B b;
} TPM2B_NAME;

typedef struct
{  // (Part 2: Structures)
    UINT8 sizeofSelect;
    BYTE  pcrSelect[PCR_SELECT_MAX];
} TPMS_PCR_SELECT;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_HASH hash;
    UINT8         sizeofSelect;
    BYTE          pcrSelect[PCR_SELECT_MAX];
} TPMS_PCR_SELECTION;

typedef struct
{  // (Part 2: Structures)
    TPM_ST            tag;
    TPMI_RH_HIERARCHY hierarchy;
```

```
    TPM2B_DIGEST      digest;
} TPMT_TK_CREATION;

typedef struct
{  // (Part 2: Structures)
    TPM_ST            tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST      digest;
} TPMT_TK_VERIFIED;

typedef struct
{  // (Part 2: Structures)
    TPM_ST            tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST      digest;
} TPMT_TK_AUTH;

typedef struct
{  // (Part 2: Structures)
    TPM_ST            tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST      digest;
} TPMT_TK_HASHCHECK;

typedef struct
{  // (Part 2: Structures)
    TPM_ALG_ID     alg;
    TPMA_ALGORITHM algProperties;
} TPMS_ALG_PROPERTY;

typedef struct
{  // (Part 2: Structures)
    TPM_PT property;
    UINT32 value;
} TPMS_TAGGED_PROPERTY;

typedef struct
{  // (Part 2: Structures)
    TPM_PT_PCR tag;
    UINT8      sizeofSelect;
    BYTE       pcrSelect[PCR_SELECT_MAX];
} TPMS_TAGGED_PCR_SELECT;

typedef struct
{  // (Part 2: Structures)
    TPM_HANDLE handle;
    TPMT_HA    policyHash;
} TPMS_TAGGED_POLICY;

typedef struct
{  // (Part 2: Structures)
    TPM_HANDLE handle;
    UINT32     timeout;
    TPMA_ACT   attributes;
} TPMS_ACT_DATA;

typedef struct
{  // (Part 2: Structures)
    UINT32 count;
    TPM_CC commandCodes[MAX_CAP_CC];
} TPML_CC;

typedef struct
{  // (Part 2: Structures)
    UINT32  count;
    TPMA_CC commandAttributes[MAX_CAP_CC];
```

```
} TPML_CCA;

typedef struct
{  // (Part 2: Structures)
    UINT32     count;
    TPM_ALG_ID algorithms[MAX_ALG_LIST_SIZE];
} TPML_ALG;

typedef struct
{  // (Part 2: Structures)
    UINT32     count;
    TPM_HANDLE handle[MAX_CAP_HANDLES];
} TPML_HANDLE;

typedef struct
{  // (Part 2: Structures)
    UINT32       count;
    TPM2B_DIGEST digests[8];
} TPML_DIGEST;

typedef struct
{  // (Part 2: Structures)
    UINT32  count;
    TPMT_HA digests[HASH_COUNT];
} TPML_DIGEST_VALUES;

typedef struct
{  // (Part 2: Structures)
    UINT32             count;
    TPMS_PCR_SELECTION pcrSelections[HASH_COUNT];
} TPML_PCR_SELECTION;

typedef struct
{  // (Part 2: Structures)
    UINT32            count;
    TPMS_ALG_PROPERTY algProperties[MAX_CAP_ALGS];
} TPML_ALG_PROPERTY;

typedef struct
{  // (Part 2: Structures)
    UINT32               count;
    TPMS_TAGGED_PROPERTY tpmProperty[MAX_TPM_PROPERTIES];
} TPML_TAGGED_TPM_PROPERTY;

typedef struct
{  // (Part 2: Structures)
    UINT32                 count;
    TPMS_TAGGED_PCR_SELECT pcrProperty[MAX_PCR_PROPERTIES];
} TPML_TAGGED_PCR_PROPERTY;

typedef struct
{  // (Part 2: Structures)
    UINT32        count;
    TPM_ECC_CURVE eccCurves[MAX_ECC_CURVES];
} TPML_ECC_CURVE;

typedef struct
{  // (Part 2: Structures)
    UINT32             count;
    TPMS_TAGGED_POLICY policies[MAX_TAGGED_POLICIES];
} TPML_TAGGED_POLICY;

typedef struct
{  // (Part 2: Structures)
    UINT32        count;
    TPMS_ACT_DATA actData[MAX_ACT_DATA];
```

```
} TPML_ACT_DATA;

typedef struct
{  // (Part 2: Structures)
    UINT32                count;
    TPM2B_VENDOR_PROPERTY vendorData[MAX_VENDOR_PROPERTY];
} TPML_VENDOR_PROPERTY;

typedef union
{  // (Part 2: Structures)
    TPML_ALG_PROPERTY        algorithms;
    TPML_HANDLE              handles;
    TPML_CCA                 command;
    TPML_CC                  ppCommands;
    TPML_CC                  auditCommands;
    TPML_PCR_SELECTION       assignedPCR;
    TPML_TAGGED_TPM_PROPERTY tpmProperties;
    TPML_TAGGED_PCR_PROPERTY pcrProperties;
#if ALG_ECC
    TPML_ECC_CURVE eccCurves;
#endif  // ALG_ECC
    TPML_TAGGED_POLICY authPolicies;
    TPML_ACT_DATA      actData;
} TPMU_CAPABILITIES;

typedef struct
{  // (Part 2: Structures)
    TPM_CAP           capability;
    TPMU_CAPABILITIES data;
} TPMS_CAPABILITY_DATA;

typedef union
{  // (Part 2: Structures)
    // NOTE: No settable capabilities are implemented in this reference code.
    UINT32 reserved;  // some compilers don't support empty unions in C
} TPMU_SET_CAPABILITIES;

typedef struct
{  // (Part 2: Structures)
    TPM_CAP               setCapability;
    TPMU_SET_CAPABILITIES data;
} TPMS_SET_CAPABILITY_DATA;

typedef struct
{  // (Part 2: Structures)
    UINT16                   size;
    TPMS_SET_CAPABILITY_DATA setCapabilityData;
} TPM2B_SET_CAPABILITY_DATA;

typedef struct
{  // (Part 2: Structures)
    UINT64      clock;
    UINT32      resetCount;
    UINT32      restartCount;
    TPMI_YES_NO safe;
} TPMS_CLOCK_INFO;

typedef struct
{  // (Part 2: Structures)
    UINT64          time;
    TPMS_CLOCK_INFO clockInfo;
} TPMS_TIME_INFO;

typedef struct
{  // (Part 2: Structures)
    TPMS_TIME_INFO time;
```

```
    UINT64         firmwareVersion;
} TPMS_TIME_ATTEST_INFO;

typedef struct
{  // (Part 2: Structures)
    TPM2B_NAME name;
    TPM2B_NAME qualifiedName;
} TPMS_CERTIFY_INFO;

typedef struct
{  // (Part 2: Structures)
    TPML_PCR_SELECTION pcrSelect;
    TPM2B_DIGEST       pcrDigest;
} TPMS_QUOTE_INFO;

typedef struct
{  // (Part 2: Structures)
    UINT64       auditCounter;
    TPM_ALG_ID   digestAlg;
    TPM2B_DIGEST auditDigest;
    TPM2B_DIGEST commandDigest;
} TPMS_COMMAND_AUDIT_INFO;

typedef struct
{  // (Part 2: Structures)
    TPMI_YES_NO  exclusiveSession;
    TPM2B_DIGEST sessionDigest;
} TPMS_SESSION_AUDIT_INFO;

typedef struct
{  // (Part 2: Structures)
    TPM2B_NAME   objectName;
    TPM2B_DIGEST creationHash;
} TPMS_CREATION_INFO;

typedef struct
{  // (Part 2: Structures)
    TPM2B_NAME          indexName;
    UINT16              offset;
    TPM2B_MAX_NV_BUFFER nvContents;
} TPMS_NV_CERTIFY_INFO;

typedef struct
{  // (Part 2: Structures)
    TPM2B_NAME   indexName;
    TPM2B_DIGEST nvDigest;
} TPMS_NV_DIGEST_CERTIFY_INFO;

typedef TPM_ST TPMI_ST_ATTEST;  // (Part 2: Structures)
typedef union
{  // (Part 2: Structures)
    TPMS_CERTIFY_INFO           certify;
    TPMS_CREATION_INFO          creation;
    TPMS_QUOTE_INFO             quote;
    TPMS_COMMAND_AUDIT_INFO     commandAudit;
    TPMS_SESSION_AUDIT_INFO     sessionAudit;
    TPMS_TIME_ATTEST_INFO       time;
    TPMS_NV_CERTIFY_INFO        nv;
    TPMS_NV_DIGEST_CERTIFY_INFO nvDigest;
} TPMU_ATTEST;

typedef struct
{  // (Part 2: Structures)
    TPM_CONSTANTS32 magic;
    TPMI_ST_ATTEST  type;
    TPM2B_NAME      qualifiedSigner;
```

```
    TPM2B_DATA      extraData;
    TPMS_CLOCK_INFO clockInfo;
    UINT64          firmwareVersion;
    TPMU_ATTEST     attested;
} TPMS_ATTEST;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   attestationData[sizeof(TPMS_ATTEST)];
    } t;
    TPM2B b;
} TPM2B_ATTEST;

typedef struct
{  // (Part 2: Structures)
    TPMI_SH_AUTH_SESSION sessionHandle;
    TPM2B_NONCE          nonce;
    TPMA_SESSION         sessionAttributes;
    TPM2B_AUTH           hmac;
} TPMS_AUTH_COMMAND;

typedef struct
{  // (Part 2: Structures)
    TPM2B_NONCE  nonce;
    TPMA_SESSION sessionAttributes;
    TPM2B_AUTH   hmac;
} TPMS_AUTH_RESPONSE;

typedef TPM_KEY_BITS TPMI_AES_KEY_BITS;       // (Part 2: Structures)
typedef TPM_KEY_BITS TPMI_SM4_KEY_BITS;       // (Part 2: Structures)
typedef TPM_KEY_BITS TPMI_CAMELLIA_KEY_BITS;  // (Part 2: Structures)
typedef union
{  // (Part 2: Structures)
#if ALG_AES
    TPMI_AES_KEY_BITS aes;
#endif  // ALG_AES
#if ALG_SM4
    TPMI_SM4_KEY_BITS sm4;
#endif  // ALG_SM4
#if ALG_CAMELLIA
    TPMI_CAMELLIA_KEY_BITS camellia;
#endif  // ALG_CAMELLIA
    TPM_KEY_BITS sym;
#if ALG_XOR
    TPMI_ALG_HASH xor ;
#endif  // ALG_XOR
} TPMU_SYM_KEY_BITS;

typedef union
{  // (Part 2: Structures)
#if ALG_AES
    TPMI_ALG_SYM_MODE aes;
#endif  // ALG_AES
#if ALG_SM4
    TPMI_ALG_SYM_MODE sm4;
#endif  // ALG_SM4
#if ALG_CAMELLIA
    TPMI_ALG_SYM_MODE camellia;
#endif  // ALG_CAMELLIA
    TPMI_ALG_SYM_MODE sym;
} TPMU_SYM_MODE;

typedef struct
```

```
{  // (Part 2: Structures)
    TPMI_ALG_SYM      algorithm;
    TPMU_SYM_KEY_BITS keyBits;
    TPMU_SYM_MODE     mode;
} TPMT_SYM_DEF;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_SYM_OBJECT algorithm;
    TPMU_SYM_KEY_BITS   keyBits;
    TPMU_SYM_MODE       mode;
} TPMT_SYM_DEF_OBJECT;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[MAX_SYM_KEY_BYTES];
    } t;
    TPM2B b;
} TPM2B_SYM_KEY;

typedef struct
{  // (Part 2: Structures)
    TPMT_SYM_DEF_OBJECT sym;
} TPMS_SYMCIPHER_PARMS;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[LABEL_MAX_BUFFER];
    } t;
    TPM2B b;
} TPM2B_LABEL;

typedef struct
{  // (Part 2: Structures)
    TPM2B_LABEL label;
    TPM2B_LABEL context;
} TPMS_DERIVE;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(TPMS_DERIVE)];
    } t;
    TPM2B b;
} TPM2B_DERIVE;

typedef union
{  // (Part 2: Structures)
    BYTE        create[MAX_SYM_DATA];
    TPMS_DERIVE derive;
} TPMU_SENSITIVE_CREATE;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(TPMU_SENSITIVE_CREATE)];
```

```
    } t;
    TPM2B b;
} TPM2B_SENSITIVE_DATA;

typedef struct
{  // (Part 2: Structures)
    TPM2B_AUTH           userAuth;
    TPM2B_SENSITIVE_DATA data;
} TPMS_SENSITIVE_CREATE;

typedef struct
{  // (Part 2: Structures)
    UINT16                size;
    TPMS_SENSITIVE_CREATE sensitive;
} TPM2B_SENSITIVE_CREATE;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_HASH hashAlg;
} TPMS_SCHEME_HASH;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_HASH hashAlg;
    UINT16        count;
} TPMS_SCHEME_ECDAA;

typedef TPM_ALG_ID TPMI_ALG_KEYEDHASH_SCHEME;  // (Part 2: Structures)

// Table "Definition of Types for HMAC_SIG_SCHEME" (Part 2: Structures)
typedef TPMS_SCHEME_HASH TPMS_SCHEME_HMAC;
#define TYPE_OF_TPMS_SCHEME_HMAC TPMS_SCHEME_HASH

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_HASH hashAlg;
    TPMI_ALG_KDF  kdf;
} TPMS_SCHEME_XOR;

typedef union
{  // (Part 2: Structures)
#if ALG_HMAC
    TPMS_SCHEME_HMAC hmac;
#endif  // ALG_HMAC
#if ALG_XOR
    TPMS_SCHEME_XOR xor ;
#endif  // ALG_XOR
} TPMU_SCHEME_KEYEDHASH;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_KEYEDHASH_SCHEME scheme;
    TPMU_SCHEME_KEYEDHASH     details;
} TPMT_KEYEDHASH_SCHEME;

// Table "Definition of Types for RSA Signature Schemes" (Part 2: Structures)
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSAPSS;
#define TYPE_OF_TPMS_SIG_SCHEME_RSAPSS TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSASSA;
#define TYPE_OF_TPMS_SIG_SCHEME_RSASSA TPMS_SCHEME_HASH

// Table "Definition of Types for ECC Signature Schemes" (Part 2: Structures)
typedef TPMS_SCHEME_ECDAA TPMS_SIG_SCHEME_ECDAA;
#define TYPE_OF_TPMS_SIG_SCHEME_ECDAA TPMS_SCHEME_ECDAA
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECDSA;
#define TYPE_OF_TPMS_SIG_SCHEME_ECDSA TPMS_SCHEME_HASH
```

```
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECSCHNORR;
#define TYPE_OF_TPMS_SIG_SCHEME_ECSCHNORR TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_EDDSA;
#define TYPE_OF_TPMS_SIG_SCHEME_EDDSA TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_EDDSA_PH;
#define TYPE_OF_TPMS_SIG_SCHEME_EDDSA_PH TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_SM2;
#define TYPE_OF_TPMS_SIG_SCHEME_SM2 TPMS_SCHEME_HASH

typedef union
{  // (Part 2: Structures)
#if ALG_HMAC
    TPMS_SCHEME_HMAC hmac;
#endif  // ALG_HMAC
#if ALG_RSASSA
    TPMS_SIG_SCHEME_RSASSA rsassa;
#endif  // ALG_RSASSA
#if ALG_RSAPSS
    TPMS_SIG_SCHEME_RSAPSS rsapss;
#endif  // ALG_RSAPSS
#if ALG_ECDSA
    TPMS_SIG_SCHEME_ECDSA ecdsa;
#endif  // ALG_ECDSA
#if ALG_ECDAA
    TPMS_SIG_SCHEME_ECDAA ecdaa;
#endif  // ALG_ECDAA
#if ALG_SM2
    TPMS_SIG_SCHEME_SM2 sm2;
#endif  // ALG_SM2
#if ALG_ECSCHNORR
    TPMS_SIG_SCHEME_ECSCHNORR ecschnorr;
#endif  // ALG_ECSCHNORR
#if ALG_EDDSA
    TPMS_SIG_SCHEME_EDDSA eddsa;
#endif  // ALG_EDDSA
#if ALG_EDDSA_PH
    TPMS_SIG_SCHEME_EDDSA_PH eddsa_ph;
#endif  // ALG_EDDSA_PH
#if ALG_LMS
    TPMS_SIG_SCHEME_LMS lms;
#endif  // ALG_LMS
#if ALG_XMSS
    TPMS_SIG_SCHEME_XMSS xmss;
#endif  // ALG_XMSS
    TPMS_SCHEME_HASH any;
} TPMU_SIG_SCHEME;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_SIG_SCHEME scheme;
    TPMU_SIG_SCHEME     details;
} TPMT_SIG_SCHEME;

// Table "Definition of Types for Encryption Schemes" (Part 2: Structures)
typedef TPMS_EMPTY TPMS_ENC_SCHEME_RSAES;
#define TYPE_OF_TPMS_ENC_SCHEME_RSAES TPMS_EMPTY
typedef TPMS_SCHEME_HASH TPMS_ENC_SCHEME_OAEP;
#define TYPE_OF_TPMS_ENC_SCHEME_OAEP TPMS_SCHEME_HASH

// Table "Definition of Types for ECC Key Exchange" (Part 2: Structures)
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECDH;
#define TYPE_OF_TPMS_KEY_SCHEME_ECDH TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECMQV;
#define TYPE_OF_TPMS_KEY_SCHEME_ECMQV TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_SM2;
#define TYPE_OF_TPMS_KEY_SCHEME_SM2 TPMS_SCHEME_HASH
```

```
// Table "Definition of Types for KDF Schemes" (Part 2: Structures)
typedef TPMS_SCHEME_HASH TPMS_KDF_SCHEME_KDF1_SP800_108;
#define TYPE_OF_TPMS_KDF_SCHEME_KDF1_SP800_108 TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_KDF_SCHEME_KDF1_SP800_56A;
#define TYPE_OF_TPMS_KDF_SCHEME_KDF1_SP800_56A TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_KDF_SCHEME_KDF2;
#define TYPE_OF_TPMS_KDF_SCHEME_KDF2 TPMS_SCHEME_HASH
typedef TPMS_SCHEME_HASH TPMS_KDF_SCHEME_MGF1;
#define TYPE_OF_TPMS_KDF_SCHEME_MGF1 TPMS_SCHEME_HASH

typedef union
{  // (Part 2: Structures)
    TPMS_SCHEME_HASH anyKdf;
#if ALG_MGF1
    TPMS_KDF_SCHEME_MGF1 mgf1;
#endif  // ALG_MGF1
#if ALG_KDF1_SP800_56A
    TPMS_KDF_SCHEME_KDF1_SP800_56A kdf1_sp800_56a;
#endif  // ALG_KDF1_SP800_56A
#if ALG_KDF2
    TPMS_KDF_SCHEME_KDF2 kdf2;
#endif  // ALG_KDF2
#if ALG_KDF1_SP800_108
    TPMS_KDF_SCHEME_KDF1_SP800_108 kdf1_sp800_108;
#endif  // ALG_KDF1_SP800_108
} TPMU_KDF_SCHEME;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_KDF    scheme;
    TPMU_KDF_SCHEME details;
} TPMT_KDF_SCHEME;

typedef TPM_ALG_ID TPMI_ALG_ASYM_SCHEME;  // (Part 2: Structures)
typedef union
{  // (Part 2: Structures)
    TPMS_SCHEME_HASH anySig;
#if ALG_RSASSA
    TPMS_SIG_SCHEME_RSASSA rsassa;
#endif  // ALG_RSASSA
#if ALG_RSAES
    TPMS_ENC_SCHEME_RSAES rsaes;
#endif  // ALG_RSAES
#if ALG_RSAPSS
    TPMS_SIG_SCHEME_RSAPSS rsapss;
#endif  // ALG_RSAPSS
#if ALG_OAEP
    TPMS_ENC_SCHEME_OAEP oaep;
#endif  // ALG_OAEP
#if ALG_ECDSA
    TPMS_SIG_SCHEME_ECDSA ecdsa;
#endif  // ALG_ECDSA
#if ALG_ECDH
    TPMS_KEY_SCHEME_ECDH ecdh;
#endif  // ALG_ECDH
#if ALG_ECDAA
    TPMS_SIG_SCHEME_ECDAA ecdaa;
#endif  // ALG_ECDAA
#if ALG_SM2
    TPMS_KEY_SCHEME_SM2 sm2;
#endif  // ALG_SM2
#if ALG_ECSCHNORR
    TPMS_SIG_SCHEME_ECSCHNORR ecschnorr;
#endif  // ALG_ECSCHNORR
#if ALG_ECMQV
```

```
    TPMS_KEY_SCHEME_ECMQV ecmqv;
#endif  // ALG_ECMQV
#if ALG_EDDSA
    TPMS_SIG_SCHEME_EDDSA eddsa;
#endif  // ALG_EDDSA
#if ALG_EDDSA_PH
    TPMS_SIG_SCHEME_EDDSA_PH eddsa_ph;
#endif  // ALG_EDDSA_PH
#if ALG_LMS
    TPMS_SIG_SCHEME_LMS lms;
#endif  // ALG_LMS
#if ALG_XMSS
    TPMS_SIG_SCHEME_XMSS xmss;
#endif  // ALG_XMSS
} TPMU_ASYM_SCHEME;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_ASYM_SCHEME scheme;
    TPMU_ASYM_SCHEME     details;
} TPMT_ASYM_SCHEME;

typedef TPM_ALG_ID TPMI_ALG_RSA_SCHEME;  // (Part 2: Structures)
typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_RSA_SCHEME scheme;
    TPMU_ASYM_SCHEME    details;
} TPMT_RSA_SCHEME;

typedef TPM_ALG_ID TPMI_ALG_RSA_DECRYPT;  // (Part 2: Structures)
typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_RSA_DECRYPT scheme;
    TPMU_ASYM_SCHEME     details;
} TPMT_RSA_DECRYPT;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[MAX_RSA_KEY_BYTES];
    } t;
    TPM2B b;
} TPM2B_PUBLIC_KEY_RSA;

typedef TPM_KEY_BITS TPMI_RSA_KEY_BITS;  // (Part 2: Structures)
typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[RSA_PRIVATE_SIZE];
    } t;
    TPM2B b;
} TPM2B_PRIVATE_KEY_RSA;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[MAX_ECC_KEY_BYTES];
    } t;
    TPM2B b;
} TPM2B_ECC_PARAMETER;
```

```
typedef struct
{  // (Part 2: Structures)
    TPM2B_ECC_PARAMETER x;
    TPM2B_ECC_PARAMETER y;
} TPMS_ECC_POINT;

typedef struct
{  // (Part 2: Structures)
    UINT16         size;
    TPMS_ECC_POINT point;
} TPM2B_ECC_POINT;

typedef TPM_ALG_ID    TPMI_ALG_ECC_SCHEME;  // (Part 2: Structures)
typedef TPM_ECC_CURVE TPMI_ECC_CURVE;       // (Part 2: Structures)
typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_ECC_SCHEME scheme;
    TPMU_ASYM_SCHEME    details;
} TPMT_ECC_SCHEME;

typedef struct
{  // (Part 2: Structures)
    TPM_ECC_CURVE       curveID;
    UINT16              keySize;
    TPMT_KDF_SCHEME     kdf;
    TPMT_ECC_SCHEME     sign;
    TPM2B_ECC_PARAMETER p;
    TPM2B_ECC_PARAMETER a;
    TPM2B_ECC_PARAMETER b;
    TPM2B_ECC_PARAMETER gX;
    TPM2B_ECC_PARAMETER gY;
    TPM2B_ECC_PARAMETER n;
    TPM2B_ECC_PARAMETER h;
} TPMS_ALGORITHM_DETAIL_ECC;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_HASH        hash;
    TPM2B_PUBLIC_KEY_RSA sig;
} TPMS_SIGNATURE_RSA;

// Table "Definition of Types for Signature" (Part 2: Structures)
typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSAPSS;
#define TYPE_OF_TPMS_SIGNATURE_RSAPSS TPMS_SIGNATURE_RSA
typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSASSA;
#define TYPE_OF_TPMS_SIGNATURE_RSASSA TPMS_SIGNATURE_RSA

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_HASH       hash;
    TPM2B_ECC_PARAMETER signatureR;
    TPM2B_ECC_PARAMETER signatureS;
} TPMS_SIGNATURE_ECC;

// Table "Definition of Types for TPMS_SIGNATURE_ECC" (Part 2: Structures)
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDAA;
#define TYPE_OF_TPMS_SIGNATURE_ECDAA TPMS_SIGNATURE_ECC
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDSA;
#define TYPE_OF_TPMS_SIGNATURE_ECDSA TPMS_SIGNATURE_ECC
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECSCHNORR;
#define TYPE_OF_TPMS_SIGNATURE_ECSCHNORR TPMS_SIGNATURE_ECC
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_EDDSA;
#define TYPE_OF_TPMS_SIGNATURE_EDDSA TPMS_SIGNATURE_ECC
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_EDDSA_PH;
#define TYPE_OF_TPMS_SIGNATURE_EDDSA_PH TPMS_SIGNATURE_ECC
```

```
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_SM2;
#define TYPE_OF_TPMS_SIGNATURE_SM2 TPMS_SIGNATURE_ECC

typedef union
{  // (Part 2: Structures)
#if ALG_HMAC
    TPMT_HA hmac;
#endif  // ALG_HMAC
#if ALG_RSASSA
    TPMS_SIGNATURE_RSASSA rsassa;
#endif  // ALG_RSASSA
#if ALG_RSAPSS
    TPMS_SIGNATURE_RSAPSS rsapss;
#endif  // ALG_RSAPSS
#if ALG_ECDSA
    TPMS_SIGNATURE_ECDSA ecdsa;
#endif  // ALG_ECDSA
#if ALG_ECDAA
    TPMS_SIGNATURE_ECDAA ecdaa;
#endif  // ALG_ECDAA
#if ALG_SM2
    TPMS_SIGNATURE_SM2 sm2;
#endif  // ALG_SM2
#if ALG_ECSCHNORR
    TPMS_SIGNATURE_ECSCHNORR ecschnorr;
#endif  // ALG_ECSCHNORR
#if ALG_EDDSA
    TPMS_SIGNATURE_EDDSA eddsa;
#endif  // ALG_EDDSA
#if ALG_EDDSA_PH
    TPMS_SIGNATURE_EDDSA_PH eddsa_ph;
#endif  // ALG_EDDSA_PH
#if ALG_LMS
    TPMS_SIGNATURE_LMS lms;
#endif  // ALG_LMS
#if ALG_XMSS
    TPMS_SIGNATURE_XMSS xmss;
#endif  // ALG_XMSS
    TPMS_SCHEME_HASH any;
} TPMU_SIGNATURE;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_SIG_SCHEME sigAlg;
    TPMU_SIGNATURE      signature;
} TPMT_SIGNATURE;

typedef union
{  // (Part 2: Structures)
#if ALG_ECC
    BYTE ecc[sizeof(TPMS_ECC_POINT)];
#endif  // ALG_ECC
#if ALG_RSA
    BYTE rsa[MAX_RSA_KEY_BYTES];
#endif  // ALG_RSA
#if ALG_SYMCIPHER
    BYTE symmetric[sizeof(TPM2B_DIGEST)];
#endif  // ALG_SYMCIPHER
#if ALG_KEYEDHASH
    BYTE keyedHash[sizeof(TPM2B_DIGEST)];
#endif  // ALG_KEYEDHASH
} TPMU_ENCRYPTED_SECRET;

typedef union
{  // (Part 2: Structures)
    struct
```

```
    {
        UINT16 size;
        BYTE   secret[sizeof(TPMU_ENCRYPTED_SECRET)];
    } t;
    TPM2B b;
} TPM2B_ENCRYPTED_SECRET;

typedef TPM_ALG_ID TPMI_ALG_PUBLIC;  // (Part 2: Structures)
typedef union
{  // (Part 2: Structures)
#if ALG_KEYEDHASH
    TPM2B_DIGEST keyedHash;
#endif  // ALG_KEYEDHASH
#if ALG_SYMCIPHER
    TPM2B_DIGEST sym;
#endif  // ALG_SYMCIPHER
#if ALG_RSA
    TPM2B_PUBLIC_KEY_RSA rsa;
#endif  // ALG_RSA
#if ALG_ECC
    TPMS_ECC_POINT ecc;
#endif  // ALG_ECC
    TPMS_DERIVE derive;
} TPMU_PUBLIC_ID;

typedef struct
{  // (Part 2: Structures)
    TPMT_KEYEDHASH_SCHEME scheme;
} TPMS_KEYEDHASH_PARMS;

typedef struct
{  // (Part 2: Structures)
    TPMT_SYM_DEF_OBJECT symmetric;
    TPMT_ASYM_SCHEME    scheme;
} TPMS_ASYM_PARMS;

typedef struct
{  // (Part 2: Structures)
    TPMT_SYM_DEF_OBJECT symmetric;
    TPMT_RSA_SCHEME     scheme;
    TPMI_RSA_KEY_BITS   keyBits;
    UINT32              exponent;
} TPMS_RSA_PARMS;

typedef struct
{  // (Part 2: Structures)
    TPMT_SYM_DEF_OBJECT symmetric;
    TPMT_ECC_SCHEME     scheme;
    TPMI_ECC_CURVE      curveID;
    TPMT_KDF_SCHEME     kdf;
} TPMS_ECC_PARMS;

typedef union
{  // (Part 2: Structures)
#if ALG_KEYEDHASH
    TPMS_KEYEDHASH_PARMS keyedHashDetail;
#endif  // ALG_KEYEDHASH
#if ALG_SYMCIPHER
    TPMS_SYMCIPHER_PARMS symDetail;
#endif  // ALG_SYMCIPHER
#if ALG_RSA
    TPMS_RSA_PARMS rsaDetail;
#endif  // ALG_RSA
#if ALG_ECC
    TPMS_ECC_PARMS eccDetail;
#endif  // ALG_ECC
```

```
    TPMS_ASYM_PARMS asymDetail;
} TPMU_PUBLIC_PARMS;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_PUBLIC   type;
    TPMU_PUBLIC_PARMS parameters;
} TPMT_PUBLIC_PARMS;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_PUBLIC   type;
    TPMI_ALG_HASH     nameAlg;
    TPMA_OBJECT       objectAttributes;
    TPM2B_DIGEST      authPolicy;
    TPMU_PUBLIC_PARMS parameters;
    TPMU_PUBLIC_ID    unique;
} TPMT_PUBLIC;

typedef struct
{  // (Part 2: Structures)
    UINT16      size;
    TPMT_PUBLIC publicArea;
} TPM2B_PUBLIC;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(TPMT_PUBLIC)];
    } t;
    TPM2B b;
} TPM2B_TEMPLATE;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[PRIVATE_VENDOR_SPECIFIC_BYTES];
    } t;
    TPM2B b;
} TPM2B_PRIVATE_VENDOR_SPECIFIC;

typedef union
{  // (Part 2: Structures)
#if ALG_RSA
    TPM2B_PRIVATE_KEY_RSA rsa;
#endif  // ALG_RSA
#if ALG_ECC
    TPM2B_ECC_PARAMETER ecc;
#endif  // ALG_ECC
#if ALG_KEYEDHASH
    TPM2B_SENSITIVE_DATA bits;
#endif  // ALG_KEYEDHASH
#if ALG_SYMCIPHER
    TPM2B_SYM_KEY sym;
#endif  // ALG_SYMCIPHER
    TPM2B_PRIVATE_VENDOR_SPECIFIC any;
} TPMU_SENSITIVE_COMPOSITE;

typedef struct
{  // (Part 2: Structures)
    TPMI_ALG_PUBLIC          sensitiveType;
    TPM2B_AUTH               authValue;
```

```
    TPM2B_DIGEST             seedValue;
    TPMU_SENSITIVE_COMPOSITE sensitive;
} TPMT_SENSITIVE;

typedef struct
{  // (Part 2: Structures)
    UINT16         size;
    TPMT_SENSITIVE sensitiveArea;
} TPM2B_SENSITIVE;

typedef struct
{  // (Part 2: Structures)
    TPM2B_DIGEST    integrityOuter;
    TPM2B_DIGEST    integrityInner;
    TPM2B_SENSITIVE sensitive;
} _PRIVATE;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(_PRIVATE)];
    } t;
    TPM2B b;
} TPM2B_PRIVATE;

typedef struct
{  // (Part 2: Structures)
    TPM2B_DIGEST integrityHMAC;
    TPM2B_DIGEST encIdentity;
} TPMS_ID_OBJECT;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   credential[sizeof(TPMS_ID_OBJECT)];
    } t;
    TPM2B b;
} TPM2B_ID_OBJECT;

// Table "Definition of TPM_NV_INDEX Bits" (Part 2: Structures)
#define TYPE_OF_TPM_NV_INDEX      UINT32
#define TPM_NV_INDEX_TO_UINT32(a) (*((UINT32*)&(a)))
#define UINT32_TO_TPM_NV_INDEX(a) (*((TPM_NV_INDEX*)&(a)))
#define TPM_NV_INDEX_TO_BYTE_ARRAY(i, a) \
    UINT32_TO_BYTE_ARRAY((TPM_NV_INDEX_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPM_NV_INDEX(i, a)      \
    {                                         \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a);   \
        i        = UINT32_TO_TPM_NV_INDEX(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned index : 24;
    unsigned RH_NV : 8;
} TPM_NV_INDEX;

// Initializer for the bit-field structure
#  define TPM_NV_INDEX_INITIALIZER(index, rh_nv) \
      {                                          \
          index, rh_nv                           \
      }
```

```
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPM_NV_INDEX Bits" (Part 2: Structures) using
bit masking
typedef UINT32 TPM_NV_INDEX;
#  define TPM_NV_INDEX_index       (TPM_NV_INDEX)(0xFFFFFF << 0)
#  define TPM_NV_INDEX_index_SHIFT 0
#  define TPM_NV_INDEX_RH_NV       (TPM_NV_INDEX)(0xFF << 24)
#  define TPM_NV_INDEX_RH_NV_SHIFT 24

//  This is the initializer for a TPM_NV_INDEX bit array.
#  define TPM_NV_INDEX_INITIALIZER(index, rh_nv) \
      (TPM_NV_INDEX)((index << 0) + (rh_nv << 24))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPM_NT Constants" (Part 2: Structures)
typedef UINT32 TPM_NT;
#define TYPE_OF_TPM_NT  UINT32
#define TPM_NT_ORDINARY (TPM_NT)(0x0)
#define TPM_NT_COUNTER  (TPM_NT)(0x1)
#define TPM_NT_BITS     (TPM_NT)(0x2)
#define TPM_NT_EXTEND   (TPM_NT)(0x4)
#define TPM_NT_PIN_FAIL (TPM_NT)(0x8)
#define TPM_NT_PIN_PASS (TPM_NT)(0x9)

typedef struct
{  // (Part 2: Structures)
    UINT32 pinCount;
    UINT32 pinLimit;
} TPMS_NV_PIN_COUNTER_PARAMETERS;

// Table "Definition of TPMA_NV Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_NV             UINT32
#define TPMA_NV_TO_UINT32(a)        (*((UINT32*)&(a)))
#define UINT32_TO_TPMA_NV(a)        (*((TPMA_NV*)&(a)))
#define TPMA_NV_TO_BYTE_ARRAY(i, a) UINT32_TO_BYTE_ARRAY((TPMA_NV_TO_UINT32(i)), (a))
#define BYTE_ARRAY_TO_TPMA_NV(i, a)         \
    {                                       \
        UINT32 x = BYTE_ARRAY_TO_UINT32(a); \
        i        = UINT32_TO_TPMA_NV(x);    \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned PPWRITE             : 1;
    unsigned OWNERWRITE          : 1;
    unsigned AUTHWRITE           : 1;
    unsigned POLICYWRITE         : 1;
    unsigned TPM_NT              : 4;
    unsigned Reserved_bits_at_8  : 2;
    unsigned POLICY_DELETE       : 1;
    unsigned WRITELOCKED         : 1;
    unsigned WRITEALL            : 1;
    unsigned WRITEDEFINE         : 1;
    unsigned WRITE_STCLEAR       : 1;
    unsigned GLOBALLOCK          : 1;
    unsigned PPREAD              : 1;
    unsigned OWNERREAD           : 1;
    unsigned AUTHREAD            : 1;
    unsigned POLICYREAD          : 1;
    unsigned Reserved_bits_at_20 : 5;
    unsigned NO_DA               : 1;
    unsigned ORDERLY             : 1;
    unsigned CLEAR_STCLEAR       : 1;
    unsigned READLOCKED          : 1;
```

```
    unsigned WRITTEN             : 1;
    unsigned PLATFORMCREATE      : 1;
    unsigned READ_STCLEAR        : 1;
} TPMA_NV;

// Initializer for the bit-field structure
#  define TPMA_NV_INITIALIZER(ppwrite,                                          \
                              ownerwrite,                                       \
                              authwrite,                                        \
                              policywrite,                                      \
                              tpm_nt,                                           \
                              bits_at_8,                                        \
                              policy_delete,                                    \
                              writelocked,                                      \
                              writeall,                                         \
                              writedefine,                                      \
                              write_stclear,                                    \
                              globallock,                                       \
                              ppread,                                           \
                              ownerread,                                        \
                              authread,                                         \
                              policyread,                                       \
                              bits_at_20,                                       \
                              no_da,                                            \
                              orderly,                                          \
                              clear_stclear,                                    \
                              readlocked,                                       \
                              written,                                          \
                              platformcreate,                                   \
                              read_stclear)                                     \
      {                                                                         \
          ppwrite, ownerwrite, authwrite, policywrite, tpm_nt, bits_at_8,       \
              policy_delete, writelocked, writeall, writedefine, write_stclear, \
              globallock, ppread, ownerread, authread, policyread, bits_at_20,  \
              no_da, orderly, clear_stclear, readlocked, written, platformcreate, \
              read_stclear                                                      \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_NV Bits" (Part 2: Structures) using bit
masking
typedef UINT32 TPMA_NV;
#  define TPMA_NV_PPWRITE        (TPMA_NV)(1 << 0)
#  define TPMA_NV_OWNERWRITE     (TPMA_NV)(1 << 1)
#  define TPMA_NV_AUTHWRITE      (TPMA_NV)(1 << 2)
#  define TPMA_NV_POLICYWRITE    (TPMA_NV)(1 << 3)
#  define TPMA_NV_TPM_NT         (TPMA_NV)(0xF << 4)
#  define TPMA_NV_TPM_NT_SHIFT   4
#  define TPMA_NV_POLICY_DELETE  (TPMA_NV)(1 << 10)
#  define TPMA_NV_WRITELOCKED    (TPMA_NV)(1 << 11)
#  define TPMA_NV_WRITEALL       (TPMA_NV)(1 << 12)
#  define TPMA_NV_WRITEDEFINE    (TPMA_NV)(1 << 13)
#  define TPMA_NV_WRITE_STCLEAR  (TPMA_NV)(1 << 14)
#  define TPMA_NV_GLOBALLOCK     (TPMA_NV)(1 << 15)
#  define TPMA_NV_PPREAD         (TPMA_NV)(1 << 16)
#  define TPMA_NV_OWNERREAD      (TPMA_NV)(1 << 17)
#  define TPMA_NV_AUTHREAD       (TPMA_NV)(1 << 18)
#  define TPMA_NV_POLICYREAD     (TPMA_NV)(1 << 19)
#  define TPMA_NV_NO_DA          (TPMA_NV)(1 << 25)
#  define TPMA_NV_ORDERLY        (TPMA_NV)(1 << 26)
#  define TPMA_NV_CLEAR_STCLEAR  (TPMA_NV)(1 << 27)
#  define TPMA_NV_READLOCKED     (TPMA_NV)(1 << 28)
#  define TPMA_NV_WRITTEN        (TPMA_NV)(1 << 29)
#  define TPMA_NV_PLATFORMCREATE (TPMA_NV)(1 << 30)
#  define TPMA_NV_READ_STCLEAR   (TPMA_NV)(1 << 31)
```

```
//  This is the initializer for a TPMA_NV bit array.
#  define TPMA_NV_INITIALIZER(ppwrite,                                          \
                              ownerwrite,                                       \
                              authwrite,                                        \
                              policywrite,                                      \
                              tpm_nt,                                           \
                              bits_at_8,                                        \
                              policy_delete,                                    \
                              writelocked,                                      \
                              writeall,                                         \
                              writedefine,                                      \
                              write_stclear,                                    \
                              globallock,                                       \
                              ppread,                                           \
                              ownerread,                                        \
                              authread,                                         \
                              policyread,                                       \
                              bits_at_20,                                       \
                              no_da,                                            \
                              orderly,                                          \
                              clear_stclear,                                    \
                              readlocked,                                       \
                              written,                                          \
                              platformcreate,                                   \
                              read_stclear)                                     \
      (TPMA_NV)((ppwrite << 0) + (ownerwrite << 1) + (authwrite << 2)           \
                + (policywrite << 3) + (tpm_nt << 4) + (policy_delete << 10)    \
                + (writelocked << 11) + (writeall << 12) + (writedefine << 13)  \
                + (write_stclear << 14) + (globallock << 15) + (ppread << 16)   \
                + (ownerread << 17) + (authread << 18) + (policyread << 19)     \
                + (no_da << 25) + (orderly << 26) + (clear_stclear << 27)       \
                + (readlocked << 28) + (written << 29) + (platformcreate << 30) \
                + (read_stclear << 31))

#endif  // USE_BIT_FIELD_STRUCTURES

// Table "Definition of TPMA_NV_EXP Bits" (Part 2: Structures)
#define TYPE_OF_TPMA_NV_EXP      UINT64
#define TPMA_NV_EXP_TO_UINT64(a) (*((UINT64*)&(a)))
#define UINT64_TO_TPMA_NV_EXP(a) (*((TPMA_NV_EXP*)&(a)))
#define TPMA_NV_EXP_TO_BYTE_ARRAY(i, a) \
    UINT64_TO_BYTE_ARRAY((TPMA_NV_EXP_TO_UINT64(i)), (a))
#define BYTE_ARRAY_TO_TPMA_NV_EXP(i, a)      \
    {                                        \
        UINT64 x = BYTE_ARRAY_TO_UINT64(a);  \
        i        = UINT64_TO_TPMA_NV_EXP(x); \
    }
#if USE_BIT_FIELD_STRUCTURES
typedef struct
{
    unsigned TPMA_NV_PPWRITE              : 1;
    unsigned TPMA_NV_OWNERWRITE           : 1;
    unsigned TPMA_NV_AUTHWRITE            : 1;
    unsigned TPMA_NV_POLICYWRITE          : 1;
    unsigned TPM_NT                       : 4;
    unsigned Reserved_bits_at_8           : 2;
    unsigned TPMA_NV_POLICY_DELETE        : 1;
    unsigned TPMA_NV_WRITELOCKED          : 1;
    unsigned TPMA_NV_WRITEALL             : 1;
    unsigned TPMA_NV_WRITEDEFINE          : 1;
    unsigned TPMA_NV_WRITE_STCLEAR        : 1;
    unsigned TPMA_NV_GLOBALLOCK           : 1;
    unsigned TPMA_NV_PPREAD               : 1;
    unsigned TPMA_NV_OWNERREAD            : 1;
    unsigned TPMA_NV_AUTHREAD             : 1;
    unsigned TPMA_NV_POLICYREAD           : 1;
```

```
    unsigned Reserved_bits_at_20           : 5;
    unsigned TPMA_NV_NO_DA                 : 1;
    unsigned TPMA_NV_ORDERLY               : 1;
    unsigned TPMA_NV_CLEAR_STCLEAR         : 1;
    unsigned TPMA_NV_READLOCKED            : 1;
    unsigned TPMA_NV_WRITTEN               : 1;
    unsigned TPMA_NV_PLATFORMCREATE        : 1;
    unsigned TPMA_NV_READ_STCLEAR          : 1;
    unsigned TPMA_EXTERNAL_NV_ENCRYPTION   : 1;
    unsigned TPMA_EXTERNAL_NV_INTEGRITY    : 1;
    unsigned TPMA_EXTERNAL_NV_ANTIROLLBACK : 1;
    unsigned Reserved_bits_at_35           : 29;
} TPMA_NV_EXP;

// Initializer for the bit-field structure
#  define TPMA_NV_EXP_INITIALIZER(tpma_nv_ppwrite,                                \
                                  tpma_nv_ownerwrite,                             \
                                  tpma_nv_authwrite,                              \
                                  tpma_nv_policywrite,                            \
                                  tpm_nt,                                         \
                                  bits_at_8,                                      \
                                  tpma_nv_policy_delete,                          \
                                  tpma_nv_writelocked,                            \
                                  tpma_nv_writeall,                               \
                                  tpma_nv_writedefine,                            \
                                  tpma_nv_write_stclear,                          \
                                  tpma_nv_globallock,                             \
                                  tpma_nv_ppread,                                 \
                                  tpma_nv_ownerread,                              \
                                  tpma_nv_authread,                               \
                                  tpma_nv_policyread,                             \
                                  bits_at_20,                                     \
                                  tpma_nv_no_da,                                  \
                                  tpma_nv_orderly,                                \
                                  tpma_nv_clear_stclear,                          \
                                  tpma_nv_readlocked,                             \
                                  tpma_nv_written,                                \
                                  tpma_nv_platformcreate,                         \
                                  tpma_nv_read_stclear,                           \
                                  tpma_external_nv_encryption,                    \
                                  tpma_external_nv_integrity,                     \
                                  tpma_external_nv_antirollback,                  \
                                  bits_at_35)                                     \
      {                                                                           \
          tpma_nv_ppwrite, tpma_nv_ownerwrite, tpma_nv_authwrite,                 \
              tpma_nv_policywrite, tpm_nt, bits_at_8, tpma_nv_policy_delete,      \
              tpma_nv_writelocked, tpma_nv_writeall, tpma_nv_writedefine,         \
              tpma_nv_write_stclear, tpma_nv_globallock, tpma_nv_ppread,          \
              tpma_nv_ownerread, tpma_nv_authread, tpma_nv_policyread, bits_at_20,\
              tpma_nv_no_da, tpma_nv_orderly, tpma_nv_clear_stclear,              \
              tpma_nv_readlocked, tpma_nv_written, tpma_nv_platformcreate,        \
              tpma_nv_read_stclear, tpma_external_nv_encryption,                  \
              tpma_external_nv_integrity, tpma_external_nv_antirollback, bits_at_35 \
      }
#else  // USE_BIT_FIELD_STRUCTURES

// This implements Table "Definition of TPMA_NV_EXP Bits" (Part 2: Structures) using
bit masking
typedef UINT64 TPMA_NV_EXP;
#  define TPMA_NV_EXP_TPMA_NV_PPWRITE               (TPMA_NV_EXP)(1 << 0)
#  define TPMA_NV_EXP_TPMA_NV_OWNERWRITE            (TPMA_NV_EXP)(1 << 1)
#  define TPMA_NV_EXP_TPMA_NV_AUTHWRITE             (TPMA_NV_EXP)(1 << 2)
#  define TPMA_NV_EXP_TPMA_NV_POLICYWRITE           (TPMA_NV_EXP)(1 << 3)
#  define TPMA_NV_EXP_TPM_NT                        (TPMA_NV_EXP)(0xF << 4)
#  define TPMA_NV_EXP_TPM_NT_SHIFT                  4
#  define TPMA_NV_EXP_TPMA_NV_POLICY_DELETE         (TPMA_NV_EXP)(1 << 10)
```

```
#  define TPMA_NV_EXP_TPMA_NV_WRITELOCKED           (TPMA_NV_EXP)(1 << 11)
#  define TPMA_NV_EXP_TPMA_NV_WRITEALL              (TPMA_NV_EXP)(1 << 12)
#  define TPMA_NV_EXP_TPMA_NV_WRITEDEFINE           (TPMA_NV_EXP)(1 << 13)
#  define TPMA_NV_EXP_TPMA_NV_WRITE_STCLEAR         (TPMA_NV_EXP)(1 << 14)
#  define TPMA_NV_EXP_TPMA_NV_GLOBALLOCK            (TPMA_NV_EXP)(1 << 15)
#  define TPMA_NV_EXP_TPMA_NV_PPREAD                (TPMA_NV_EXP)(1 << 16)
#  define TPMA_NV_EXP_TPMA_NV_OWNERREAD             (TPMA_NV_EXP)(1 << 17)
#  define TPMA_NV_EXP_TPMA_NV_AUTHREAD              (TPMA_NV_EXP)(1 << 18)
#  define TPMA_NV_EXP_TPMA_NV_POLICYREAD            (TPMA_NV_EXP)(1 << 19)
#  define TPMA_NV_EXP_TPMA_NV_NO_DA                 (TPMA_NV_EXP)(1 << 25)
#  define TPMA_NV_EXP_TPMA_NV_ORDERLY               (TPMA_NV_EXP)(1 << 26)
#  define TPMA_NV_EXP_TPMA_NV_CLEAR_STCLEAR         (TPMA_NV_EXP)(1 << 27)
#  define TPMA_NV_EXP_TPMA_NV_READLOCKED            (TPMA_NV_EXP)(1 << 28)
#  define TPMA_NV_EXP_TPMA_NV_WRITTEN               (TPMA_NV_EXP)(1 << 29)
#  define TPMA_NV_EXP_TPMA_NV_PLATFORMCREATE        (TPMA_NV_EXP)(1 << 30)
#  define TPMA_NV_EXP_TPMA_NV_READ_STCLEAR          (TPMA_NV_EXP)(1 << 31)
#  define TPMA_NV_EXP_TPMA_EXTERNAL_NV_ENCRYPTION   (TPMA_NV_EXP)(1 << 32)
#  define TPMA_NV_EXP_TPMA_EXTERNAL_NV_INTEGRITY    (TPMA_NV_EXP)(1 << 33)
#  define TPMA_NV_EXP_TPMA_EXTERNAL_NV_ANTIROLLBACK (TPMA_NV_EXP)(1 << 34)

//  This is the initializer for a TPMA_NV_EXP bit array.
#  define TPMA_NV_EXP_INITIALIZER(tpma_nv_ppwrite,                                \
                                  tpma_nv_ownerwrite,                             \
                                  tpma_nv_authwrite,                              \
                                  tpma_nv_policywrite,                            \
                                  tpm_nt,                                         \
                                  bits_at_8,                                      \
                                  tpma_nv_policy_delete,                          \
                                  tpma_nv_writelocked,                            \
                                  tpma_nv_writeall,                               \
                                  tpma_nv_writedefine,                            \
                                  tpma_nv_write_stclear,                          \
                                  tpma_nv_globallock,                             \
                                  tpma_nv_ppread,                                 \
                                  tpma_nv_ownerread,                              \
                                  tpma_nv_authread,                               \
                                  tpma_nv_policyread,                             \
                                  bits_at_20,                                     \
                                  tpma_nv_no_da,                                  \
                                  tpma_nv_orderly,                                \
                                  tpma_nv_clear_stclear,                          \
                                  tpma_nv_readlocked,                             \
                                  tpma_nv_written,                                \
                                  tpma_nv_platformcreate,                         \
                                  tpma_nv_read_stclear,                           \
                                  tpma_external_nv_encryption,                    \
                                  tpma_external_nv_integrity,                     \
                                  tpma_external_nv_antirollback,                  \
                                  bits_at_35)                                     \
      (TPMA_NV_EXP)((tpma_nv_ppwrite << 0) + (tpma_nv_ownerwrite << 1)            \
                  + (tpma_nv_authwrite << 2) + (tpma_nv_policywrite << 3)         \
                  + (tpm_nt << 4) + (tpma_nv_policy_delete << 10)                 \
                  + (tpma_nv_writelocked << 11) + (tpma_nv_writeall << 12)        \
                  + (tpma_nv_writedefine << 13) + (tpma_nv_write_stclear << 14)   \
                  + (tpma_nv_globallock << 15) + (tpma_nv_ppread << 16)           \
                  + (tpma_nv_ownerread << 17) + (tpma_nv_authread << 18)          \
                  + (tpma_nv_policyread << 19) + (tpma_nv_no_da << 25)            \
                  + (tpma_nv_orderly << 26) + (tpma_nv_clear_stclear << 27)       \
                  + (tpma_nv_readlocked << 28) + (tpma_nv_written << 29)          \
                  + (tpma_nv_platformcreate << 30) + (tpma_nv_read_stclear << 31) \
                  + (tpma_external_nv_encryption << 32)                           \
                  + (tpma_external_nv_integrity << 33)                            \
                  + (tpma_external_nv_antirollback << 34))

#endif  // USE_BIT_FIELD_STRUCTURES
```

```
typedef struct
{  // (Part 2: Structures)
    TPMI_RH_NV_LEGACY_INDEX nvIndex;
    TPMI_ALG_HASH           nameAlg;
    TPMA_NV                 attributes;
    TPM2B_DIGEST            authPolicy;
    UINT16                  dataSize;
} TPMS_NV_PUBLIC;

typedef struct
{  // (Part 2: Structures)
    UINT16         size;
    TPMS_NV_PUBLIC nvPublic;
} TPM2B_NV_PUBLIC;

typedef struct
{  // (Part 2: Structures)
    TPMI_RH_NV_EXP_INDEX nvIndex;
    TPMI_ALG_HASH        nameAlg;
    TPMA_NV_EXP          attributes;
    TPM2B_DIGEST         authPolicy;
    UINT16               dataSize;
} TPMS_NV_PUBLIC_EXP_ATTR;

typedef union
{  // (Part 2: Structures)
    TPMS_NV_PUBLIC          nvIndex;
    TPMS_NV_PUBLIC_EXP_ATTR externalNV;
    TPMS_NV_PUBLIC          permanentNV;
} TPMU_NV_PUBLIC_2;

typedef struct
{  // (Part 2: Structures)
    TPM_HT           handleType;
    TPMU_NV_PUBLIC_2 nvPublic2;
} TPMT_NV_PUBLIC_2;

typedef struct
{  // (Part 2: Structures)
    UINT16           size;
    TPMT_NV_PUBLIC_2 nvPublic2;
} TPM2B_NV_PUBLIC_2;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[MAX_CONTEXT_SIZE];
    } t;
    TPM2B b;
} TPM2B_CONTEXT_SENSITIVE;

typedef struct
{  // (Part 2: Structures)
    TPM2B_DIGEST            integrity;
    TPM2B_CONTEXT_SENSITIVE encrypted;
} TPMS_CONTEXT_DATA;

typedef union
{  // (Part 2: Structures)
    struct
    {
        UINT16 size;
        BYTE   buffer[sizeof(TPMS_CONTEXT_DATA)];
    } t;
```

```
    TPM2B b;
} TPM2B_CONTEXT_DATA;

typedef struct
{  // (Part 2: Structures)
    UINT64              sequence;
    TPMI_DH_SAVED       savedHandle;
    TPMI_RH_HIERARCHY   hierarchy;
    TPM2B_CONTEXT_DATA contextBlob;
} TPMS_CONTEXT;

typedef struct
{  // (Part 2: Structures)
    TPML_PCR_SELECTION pcrSelect;
    TPM2B_DIGEST       pcrDigest;
    TPMA_LOCALITY      locality;
    TPM_ALG_ID         parentNameAlg;
    TPM2B_NAME         parentName;
    TPM2B_NAME         parentQualifiedName;
    TPM2B_DATA         outsideInfo;
} TPMS_CREATION_DATA;

typedef struct
{  // (Part 2: Structures)
    UINT16              size;
    TPMS_CREATION_DATA creationData;
} TPM2B_CREATION_DATA;

// Table "Definition of TPM_AT Constants" (Part 2: Structures)
typedef UINT32 TPM_AT;
#define TYPE_OF_TPM_AT UINT32
#define TPM_AT_ANY     (TPM_AT)(0x00000000)
#define TPM_AT_ERROR   (TPM_AT)(0x00000001)
#define TPM_AT_PV1     (TPM_AT)(0x00000002)
#define TPM_AT_VEND    (TPM_AT)(0x80000000)

// Table "Definition of TPM_AE Constants" (Part 2: Structures)
typedef UINT32 TPM_AE;
#define TYPE_OF_TPM_AE UINT32
#define TPM_AE_NONE    (TPM_AE)(0x00000000)

typedef struct
{  // (Part 2: Structures)
    TPM_AT tag;
    UINT32 data;
} TPMS_AC_OUTPUT;

typedef struct
{  // (Part 2: Structures)
    UINT32         count;
    TPMS_AC_OUTPUT acCapabilities[MAX_AC_CAPABILITIES];
} TPML_AC_CAPABILITIES;

#endif  // _TPM_INCLUDE_PRIVATE_TPMTYPES_H_
```

6.248 /tpm/include/public/tpm_public.h

```
#include <TpmConfiguration/TpmBuildSwitches.h>
#include <TpmConfiguration/TpmProfile.h>

#include <public/VerifyConfiguration.h>
#include <public/BaseTypes.h>
#include <public/TPMB.h>
#include <public/MinMax.h>
#include <public/tpm_radix.h>
```

```
#include <public/TpmTypes.h>
```

6.249 /tpm/include/public/tpm_radix.h

```
//** Introduction
// Common defines for supporting large numbers and cryptographic buffer sizing.
//**********************
#ifndef RADIX_BITS
#  if defined(__x86_64__) || defined(__x86_64) || defined(__amd64__)                \
      || defined(__amd64) || defined(_WIN64) || defined(_M_X64) || defined(_M_ARM64) \
      || defined(__aarch64__) || defined(__PPC64__) || defined(__s390x__)            \
      || defined(__powerpc64__) || defined(__ppc64__)
#    define RADIX_BITS 64
#  elif defined(__i386__) || defined(__i386) || defined(i386) || defined(_WIN32) \
      || defined(_M_IX86)
#    define RADIX_BITS 32
#  elif defined(_M_ARM) || defined(__arm__) || defined(__thumb__)
#    define RADIX_BITS 32
#  elif defined(__riscv)
// __riscv and __riscv_xlen are standardized by the RISC-V community and should be
available
// on any compliant compiler.
//
// https://github.com/riscv-non-isa/riscv-toolchain-conventions
#    define RADIX_BITS __riscv_xlen
#  else
#    error Unable to determine RADIX_BITS from compiler environment
#  endif
#endif  // RADIX_BITS

#if RADIX_BITS == 64
#  define RADIX_BYTES 8
#  define RADIX_LOG2  6
#elif RADIX_BITS == 32
#  define RADIX_BYTES 4
#  define RADIX_LOG2  5
#else
#  error "RADIX_BITS must either be 32 or 64"
#endif

#define HASH_ALIGNMENT      RADIX_BYTES
#define SYMMETRIC_ALIGNMENT RADIX_BYTES

#define RADIX_MOD(x) ((x) & ((1 << RADIX_LOG2) - 1))
#define RADIX_DIV(x) ((x) >> RADIX_LOG2)
#define RADIX_MASK   ((((crypt_uword_t)1) << RADIX_LOG2) - 1)

#define BITS_TO_CRYPT_WORDS(bits)   RADIX_DIV((bits) + (RADIX_BITS - 1))
#define BYTES_TO_CRYPT_WORDS(bytes) BITS_TO_CRYPT_WORDS(bytes * 8)
#define SIZE_IN_CRYPT_WORDS(thing)  BYTES_TO_CRYPT_WORDS(sizeof(thing))

#if RADIX_BITS == 64
#  define SWAP_CRYPT_WORD(x) REVERSE_ENDIAN_64(x)
typedef uint64_t crypt_uword_t;
typedef int64_t  crypt_word_t;
#  define TO_CRYPT_WORD_64              BIG_ENDIAN_BYTES_TO_UINT64
#  define TO_CRYPT_WORD_32(a, b, c, d) TO_CRYPT_WORD_64(0, 0, 0, 0, a, b, c, d)
#elif RADIX_BITS == 32
#  define SWAP_CRYPT_WORD(x) REVERSE_ENDIAN_32((x))
typedef uint32_t crypt_uword_t;
typedef int32_t  crypt_word_t;
#  define TO_CRYPT_WORD_64(a, b, c, d, e, f, g, h) \
     BIG_ENDIAN_BYTES_TO_UINT32(e, f, g, h), BIG_ENDIAN_BYTES_TO_UINT32(a, b, c, d)
#endif
```

```
#define MAX_CRYPT_UWORD (~((crypt_uword_t)0))
#define MAX_CRYPT_WORD  ((crypt_word_t)(MAX_CRYPT_UWORD >> 1))
#define MIN_CRYPT_WORD  (~MAX_CRYPT_WORD)

// Avoid expanding LARGEST_NUMBER into a long expression that inlines 3 other long
expressions.
// TODO: Decrease the size of each of the MAX_* expressions with improvements to the
code generator.
#if ALG_RSA == ALG_YES
// The smallest supported RSA key (1024 bits) is larger than
// the largest supported ECC curve (628 bits)
// or the largest supported digest (512 bits)
#  define LARGEST_NUMBER MAX_RSA_KEY_BYTES
#elif ALG_ECC == ALG_YES
#  define LARGEST_NUMBER MAX(MAX_ECC_KEY_BYTES, MAX_DIGEST_SIZE)
#else
#  define LARGEST_NUMBER MAX_DIGEST_SIZE
#endif  // ALG_RSA == YES

#define LARGEST_NUMBER_BITS (LARGEST_NUMBER * 8)

#define MAX_ECC_PARAMETER_BYTES (MAX_ECC_KEY_BYTES * ALG_ECC)

// These macros use the selected libraries to get the proper include files.
// clang-format off
#define LIB_QUOTE(_STRING_)                     #_STRING_
#define LIB_INCLUDE2(_PREFIX_, _LIB_, _TYPE_)
LIB_QUOTE(_LIB_/_PREFIX_##_LIB_##_TYPE_.h)
#define LIB_INCLUDE(_PREFIX_, _LIB_, _TYPE_)    LIB_INCLUDE2(_PREFIX_,_LIB_, _TYPE_)
// clang-format on
```

6.250 /tpm/include/public/VerifyConfiguration.h

```
//
// This verifies that information expected from the consumer's TpmConfiguration is
// set properly and consistently.
//
#ifndef _VERIFY_CONFIGURATION_H
#define _VERIFY_CONFIGURATION_H

// verify these defines are either YES or NO.
#define MUST_BE_0_OR_1(x) MUST_BE(((x) == 0) || ((x) == 1))

// Debug Options
MUST_BE_0_OR_1(DEBUG);
MUST_BE_0_OR_1(SIMULATION);
MUST_BE_0_OR_1(DRBG_DEBUG_PRINT);
MUST_BE_0_OR_1(CERTIFYX509_DEBUG);
MUST_BE_0_OR_1(USE_DEBUG_RNG);

// RSA Debug Options
MUST_BE_0_OR_1(RSA_INSTRUMENT);
MUST_BE_0_OR_1(USE_RSA_KEY_CACHE);
MUST_BE_0_OR_1(USE_KEY_CACHE_FILE);

// Test Options
MUST_BE_0_OR_1(ALLOW_FORCE_FAILURE_MODE);

// Internal checks
MUST_BE_0_OR_1(LIBRARY_COMPATIBILITY_CHECK);
MUST_BE_0_OR_1(COMPILER_CHECKS);
MUST_BE_0_OR_1(RUNTIME_SIZE_CHECKS);

// Compliance options
MUST_BE_0_OR_1(FIPS_COMPLIANT);
```

```
MUST_BE_0_OR_1(USE_SPEC_COMPLIANT_PROOFS);
MUST_BE_0_OR_1(SKIP_PROOF_ERRORS);

// Implementation alternatives - should not change external behavior
MUST_BE_0_OR_1(TABLE_DRIVEN_DISPATCH);
MUST_BE_0_OR_1(TABLE_DRIVEN_MARSHAL);
MUST_BE_0_OR_1(USE_MARSHALING_DEFINES);
MUST_BE_0_OR_1(COMPRESSED_LISTS);
MUST_BE_0_OR_1(USE_BIT_FIELD_STRUCTURES);
MUST_BE_0_OR_1(RSA_KEY_SIEVE);

// Implementation alternatives - changes external behavior
MUST_BE_0_OR_1(_DRBG_STATE_SAVE);
MUST_BE_0_OR_1(USE_DA_USED);
MUST_BE_0_OR_1(ENABLE_SELF_TESTS);
MUST_BE_0_OR_1(CLOCK_STOPS);
MUST_BE_0_OR_1(ACCUMULATE_SELF_HEAL_TIMER);
MUST_BE_0_OR_1(FAIL_TRACE);

// Vendor alternatives
// Check VENDOR_PERMANENT_AUTH_ENABLED & VENDOR_PERMANENT_AUTH_HANDLE are consistent
MUST_BE_0_OR_1(VENDOR_PERMANENT_AUTH_ENABLED);

#if VENDOR_PERMANENT_AUTH_ENABLED == YES
#  if !defined(VENDOR_PERMANENT_AUTH_HANDLE)            \
      || VENDOR_PERMANENT_AUTH_HANDLE < TPM_RH_AUTH_00 \
      || VENDOR_PERMANENT_AUTH_HANDLE > TPM_RH_AUTH_FF
#    error VENDOR_PERMANENT_AUTH_ENABLED requires a valid definition for
VENDOR_PERMANENT_AUTH_HANDLE, see Part2
#  endif
#else
#  if defined(VENDOR_PERMANENT_AUTH_HANDLE)
#    error VENDOR_PERMANENT_AUTH_HANDLE requires VENDOR_PERMANENT_AUTH_ENABLED to be
YES
#  endif
#endif

// now check for inconsistent combinations of options
#if USE_KEY_CACHE_FILE && !USE_RSA_KEY_CACHE
#  error cannot use USE_KEY_CACHE_FILE if not using USE_RSA_KEY_CACHE
#endif

#if !DEBUG
#  if USE_KEY_CACHE_FILE || USE_RSA_KEY_CACHE || DRBG_DEBUG_PRINT \
      || CERTIFYX509_DEBUG || USE_DEBUG_RNG
#    error using insecure options not in DEBUG mode.
#  endif
#endif

#if !SIMULATION
#  if USE_KEY_CACHE_FILE
#    error USE_KEY_CACHE_FILE requires SIMULATION
#  endif
#  if RSA_INSTRUMENT
#    error RSA_INSTRUMENT requires SIMULATION
#  endif
#  if USE_DEBUG_RNG
#    error USE_DEBUG_RNG requires SIMULATION
#  endif
#endif

#endif  // _VERIFY_CONFIGURATION_H
```

6.251 /tpm/include/public/prototypes/TpmFail_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Apr  2, 2019  Time: 03:18:00PM
 */

#ifndef _TPM_FAIL_FP_H_
#define _TPM_FAIL_FP_H_

//*** SetForceFailureMode()
// This function is called by the simulator to enable failure mode testing.
#if SIMULATION
LIB_EXPORT void SetForceFailureMode(void);
#endif

//*** TpmFail()
// This function is called by TPM.lib when a failure occurs. It will set up the
// failure values to be returned on TPM2_GetTestResult().
NORETURN void TpmFail(
#if FAIL_TRACE
    const char* function,
    int         line,
#else
    uint64_t locationCode,
#endif
    int failureCode);

//*** TpmFailureMode(
// This function is called by the interface code when the platform is in failure
// mode.
void TpmFailureMode(uint32_t        inRequestSize,    // IN: command buffer size
                    unsigned char*  inRequest,        // IN: command buffer
                    uint32_t*       outResponseSize,  // OUT: response buffer size
                    unsigned char** outResponse       // OUT: response buffer
);

//*** UnmarshalFail()
// This is a stub that is used to catch an attempt to unmarshal an entry
// that is not defined. Don't ever expect this to be called but...
void UnmarshalFail(void* type, BYTE** buffer, INT32* size);

#endif  // _TPM_FAIL_FP_H_
```

7 TPM Reference Implementation Source Files

7.1 /tpm/src/command/Asymmetric/ECC_Decrypt.c

```
#include "Tpm.h"
#include "ECC_Decrypt_fp.h"
#include "CryptEccCrypt_fp.h"

#if CC_ECC_Decrypt  // Conditional expansion of this file

//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES        key referenced by 'keyHandle' is restricted
//      TPM_RC_KEY                keyHandle does not reference an ECC key
//      TPM_RC_NO_RESULT        internal error in big number processing
//      TPM_RC_SCHEME            bad scheme
//      TPM_RC_VALUE            C3 did not match hash of recovered data
TPM_RC
TPM2_ECC_Decrypt(ECC_Decrypt_In*  in,  // IN: input parameter list
                 ECC_Decrypt_Out* out  // OUT: output parameter list
)
{
    OBJECT* key = HandleToObject(in->keyHandle);
    // Parameter validation
    // Must be the correct type of key with correct attributes
    if(key->publicArea.type != TPM_ALG_ECC)
        return TPM_RC_KEY + RC_ECC_Decrypt_keyHandle;
    if(IS_ATTRIBUTE(key->publicArea.objectAttributes, TPMA_OBJECT, restricted)
       || !IS_ATTRIBUTE(key->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
        return TPM_RCS_ATTRIBUTES + RC_ECC_Decrypt_keyHandle;
    // Have to have a scheme selected
    if(!CryptEccSelectScheme(key, &in->inScheme))
        return TPM_RCS_SCHEME + RC_ECC_Decrypt_inScheme;
    //  Command Output
    return CryptEccDecrypt(
        key, &in->inScheme, &out->plainText, &in->C1.point, &in->C2, &in->C3);
}

#endif  // CC_ECC_Decrypt
```

7.2 /tpm/src/command/Asymmetric/ECC_Encrypt.c

```
#include "Tpm.h"
#include "ECC_Encrypt_fp.h"

#if CC_ECC_Encrypt  // Conditional expansion of this file

//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES        key referenced by 'keyHandle' is restricted
//      TPM_RC_KEY                keyHandle does not reference an ECC key
//      TPM_RCS_SCHEME            bad scheme
TPM_RC
TPM2_ECC_Encrypt(ECC_Encrypt_In*  in,  // IN: input parameter list
                 ECC_Encrypt_Out* out  // OUT: output parameter list
)
{
    OBJECT* pubKey = HandleToObject(in->keyHandle);
    // Parameter validation
    if(pubKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RC_KEY + RC_ECC_Encrypt_keyHandle;
    // Have to have a scheme selected
    if(!CryptEccSelectScheme(pubKey, &in->inScheme))
        return TPM_RCS_SCHEME + RC_ECC_Encrypt_inScheme;
    //  Command Output
```

```
    return CryptEccEncrypt(
        pubKey, &in->inScheme, &in->plainText, &out->C1.point, &out->C2, &out->C3);
}

#endif  // CC_ECC_Encrypt
```

7.3 /tpm/src/command/Asymmetric/ECC_Parameters.c

```
#include "Tpm.h"
#include "ECC_Parameters_fp.h"

#if CC_ECC_Parameters  // Conditional expansion of this file

/*(See part 3 specification)
// This command returns the parameters of an ECC curve identified by its TCG
// assigned curveID
*/
//  Return Type: TPM_RC
//      TPM_RC_VALUE                    Unsupported ECC curve ID
TPM_RC
TPM2_ECC_Parameters(ECC_Parameters_In*  in,  // IN: input parameter list
                    ECC_Parameters_Out* out  // OUT: output parameter list
)
{
    // Command Output

    // Get ECC curve parameters
    if(CryptEccGetParameters(in->curveID, &out->parameters))
        return TPM_RC_SUCCESS;
    else
        return TPM_RCS_VALUE + RC_ECC_Parameters_curveID;
}

#endif  // CC_ECC_Parameters
```

7.4 /tpm/src/command/Asymmetric/ECDH_KeyGen.c

```
#include "Tpm.h"
#include "ECDH_KeyGen_fp.h"

#if CC_ECDH_KeyGen  // Conditional expansion of this file

/*(See part 3 specification)
// This command uses the TPM to generate an ephemeral public key and the product
// of the ephemeral private key and the public portion of an ECC key.
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY              'keyHandle' does not reference an ECC key
TPM_RC
TPM2_ECDH_KeyGen(ECDH_KeyGen_In*  in,  // IN: input parameter list
                 ECDH_KeyGen_Out* out  // OUT: output parameter list
)
{
    OBJECT*             eccKey;
    TPM2B_ECC_PARAMETER sensitive;
    TPM_RC              result;

    // Input Validation

    eccKey = HandleToObject(in->keyHandle);

    // Referenced key must be an ECC key
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RCS_KEY + RC_ECDH_KeyGen_keyHandle;
```

```
    // Command Output
    do
    {
        TPMT_PUBLIC* keyPublic = &eccKey->publicArea;
        // Create ephemeral ECC key
        result = CryptEccNewKeyPair(&out->pubPoint.point,
                                    &sensitive,
                                    keyPublic->parameters.eccDetail.curveID);
        if(result == TPM_RC_SUCCESS)
        {
            // Compute Z
            result = CryptEccPointMultiply(&out->zPoint.point,
                                           keyPublic->parameters.eccDetail.curveID,
                                           &keyPublic->unique.ecc,
                                           &sensitive,
                                           NULL,
                                           NULL);
            // The point in the key is not on the curve. Indicate
            // that the key is bad.
            if(result == TPM_RC_ECC_POINT)
                return TPM_RCS_KEY + RC_ECDH_KeyGen_keyHandle;
            // The other possible error from CryptEccPointMultiply is
            // TPM_RC_NO_RESULT indicating that the multiplication resulted in
            // the point at infinity, so get a new random key and start over
            // BTW, this never happens.
        }
    } while(result == TPM_RC_NO_RESULT);
    return result;
}

#endif  // CC_ECDH_KeyGen
```

7.5 /tpm/src/command/Asymmetric/ECDH_ZGen.c

```
#include "Tpm.h"
#include "ECDH_ZGen_fp.h"

#if CC_ECDH_ZGen  // Conditional expansion of this file

/*(See part 3 specification)
// This command uses the TPM to recover the Z value from a public point
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               key referenced by 'keyA' is restricted or
//                                      not a decrypt key
//      TPM_RC_KEY                      key referenced by 'keyA' is not an ECC key
//      TPM_RC_NO_RESULT                multiplying 'inPoint' resulted in a
//                                      point at infinity
//      TPM_RC_SCHEME                   the scheme of the key referenced by 'keyA'
//                                      is not TPM_ALG_NULL, TPM_ALG_ECDH,
TPM_RC
TPM2_ECDH_ZGen(ECDH_ZGen_In*  in,  // IN: input parameter list
               ECDH_ZGen_Out* out  // OUT: output parameter list
)
{
    TPM_RC  result;
    OBJECT* eccKey;

    // Input Validation
    eccKey = HandleToObject(in->keyHandle);

    // Selected key must be a non-restricted, decrypt ECC key
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RCS_KEY + RC_ECDH_ZGen_keyHandle;
```

```
    // Selected key needs to be unrestricted with the 'decrypt' attribute
    if(IS_ATTRIBUTE(eccKey->publicArea.objectAttributes, TPMA_OBJECT, restricted)
       || !IS_ATTRIBUTE(eccKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
        return TPM_RCS_ATTRIBUTES + RC_ECDH_ZGen_keyHandle;
    // Make sure the scheme allows this use
    if(eccKey->publicArea.parameters.eccDetail.scheme.scheme != TPM_ALG_ECDH
       && eccKey->publicArea.parameters.eccDetail.scheme.scheme != TPM_ALG_NULL)
        return TPM_RCS_SCHEME + RC_ECDH_ZGen_keyHandle;
    // Command Output
    // Compute Z. TPM_RC_ECC_POINT or TPM_RC_NO_RESULT may be returned here.
    result = CryptEccPointMultiply(&out->outPoint.point,
                                   eccKey->publicArea.parameters.eccDetail.curveID,
                                   &in->inPoint.point,
                                   &eccKey->sensitive.sensitive.ecc,
                                   NULL,
                                   NULL);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_ECDH_ZGen_inPoint);
    return result;
}

#endif  // CC_ECDH_ZGen
```

7.6 /tpm/src/command/Asymmetric/EC_Ephemeral.c

```
#include "Tpm.h"
#include "EC_Ephemeral_fp.h"

#if CC_EC_Ephemeral  // Conditional expansion of this file

/*(See part 3 specification)
// This command creates an ephemeral key using the commit mechanism
*/
//  Return Type: TPM_RC
// TPM_RC_NO_RESULT             the TPM is not able to generate an 'r' value
TPM_RC
TPM2_EC_Ephemeral(EC_Ephemeral_In*  in,  // IN: input parameter list
                  EC_Ephemeral_Out* out  // OUT: output parameter list
)
{
    TPM2B_ECC_PARAMETER r;
    TPM_RC              result;
    //
    do
    {
        // Get the random value that will be used in the point multiplications
        // Note: this does not commit the count.
        if(!CryptGenerateR(&r, NULL, in->curveID, NULL))
            return TPM_RC_NO_RESULT;
        // do a point multiply
        result =
            CryptEccPointMultiply(&out->Q.point, in->curveID, NULL, &r, NULL, NULL);
        // commit the count value if either the r value results in the point at
        // infinity or if the value is good. The commit on the r value for infinity
        // is so that the r value will be skipped.
        if((result == TPM_RC_SUCCESS) || (result == TPM_RC_NO_RESULT))
            out->counter = CryptCommit();
    } while(result == TPM_RC_NO_RESULT);

    return TPM_RC_SUCCESS;
}

#endif  // CC_EC_Ephemeral
```

7.7 /tpm/src/command/Asymmetric/RSA_Decrypt.c

```
#include "Tpm.h"
#include "RSA_Decrypt_fp.h"

#if CC_RSA_Decrypt  // Conditional expansion of this file

/*(See part 3 specification)
// decrypts the provided data block and removes the padding if applicable
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'decrypt' is not SET or if 'restricted' is SET in
//                              the key referenced by 'keyHandle'
//      TPM_RC_BINDING          The public and private parts of the key are not
//                              properly bound
//      TPM_RC_KEY              'keyHandle' does not reference an unrestricted
//                              decrypt key
//      TPM_RC_SCHEME           incorrect input scheme, or the chosen
//                              'scheme' is not a valid RSA decrypt scheme
//      TPM_RC_SIZE             'cipherText' is not the size of the modulus
//                              of key referenced by 'keyHandle'
//      TPM_RC_VALUE            'label' is not a null terminated string or the value
//                              of 'cipherText' is greater that the modulus of
//                              'keyHandle' or the encoding of the data is not
//                              valid

TPM_RC
TPM2_RSA_Decrypt(RSA_Decrypt_In*  in,  // IN: input parameter list
                 RSA_Decrypt_Out* out  // OUT: output parameter list
)
{
    TPM_RC            result;
    OBJECT*           rsaKey;
    TPMT_RSA_DECRYPT* scheme;

    // Input Validation

    rsaKey = HandleToObject(in->keyHandle);

    // The selected key must be an RSA key
    if(rsaKey->publicArea.type != TPM_ALG_RSA)
        return TPM_RCS_KEY + RC_RSA_Decrypt_keyHandle;

    // The selected key must be an unrestricted decryption key
    if(IS_ATTRIBUTE(rsaKey->publicArea.objectAttributes, TPMA_OBJECT, restricted)
       || !IS_ATTRIBUTE(rsaKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
        return TPM_RCS_ATTRIBUTES + RC_RSA_Decrypt_keyHandle;

    // NOTE: Proper operation of this command requires that the sensitive area
    // of the key is loaded. This is assured because authorization is required
    // to use the sensitive area of the key. In order to check the authorization,
    // the sensitive area has to be loaded, even if authorization is with policy.

    // If label is present, make sure that it is a NULL-terminated string
    if(!IsLabelProperlyFormatted(&in->label.b))
        return TPM_RCS_VALUE + RC_RSA_Decrypt_label;
    // Command Output
    // Select a scheme for decrypt.
    scheme = CryptRsaSelectScheme(in->keyHandle, &in->inScheme);
    if(scheme == NULL)
        return TPM_RCS_SCHEME + RC_RSA_Decrypt_inScheme;

    // Decryption.  TPM_RC_VALUE, TPM_RC_SIZE, and TPM_RC_KEY error may be
    // returned by CryptRsaDecrypt.
    // NOTE: CryptRsaDecrypt can also return TPM_RC_ATTRIBUTES or TPM_RC_BINDING
    // when the key is not a decryption key but that was checked above.
```

```
    out->message.t.size = sizeof(out->message.t.buffer);
    result              = CryptRsaDecrypt(
        &out->message.b, &in->cipherText.b, rsaKey, scheme, &in->label.b);
    return result;
}

#endif  // CC_RSA_Decrypt
```

7.8 /tpm/src/command/Asymmetric/RSA_Encrypt.c

```
#include "Tpm.h"
#include "RSA_Encrypt_fp.h"

#if CC_RSA_Encrypt  // Conditional expansion of this file

/*(See part 3 specification)
// This command performs the padding and encryption of a data block
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES           'decrypt' attribute is not SET in key referenced
//                                  by 'keyHandle'
//      TPM_RC_KEY                  'keyHandle' does not reference an RSA key
//      TPM_RC_SCHEME               incorrect input scheme, or the chosen
//                                  scheme is not a valid RSA decrypt scheme
//      TPM_RC_VALUE                the numeric value of 'message' is greater than
//                                  the public modulus of the key referenced by
//                                  'keyHandle', or 'label' is not a null-terminated
//                                  string
TPM_RC
TPM2_RSA_Encrypt(RSA_Encrypt_In*  in,  // IN: input parameter list
                 RSA_Encrypt_Out* out  // OUT: output parameter list
)
{
    TPM_RC             result;
    OBJECT*            rsaKey;
    TPMT_RSA_DECRYPT* scheme;
    // Input Validation
    rsaKey = HandleToObject(in->keyHandle);

    // selected key must be an RSA key
    if(rsaKey->publicArea.type != TPM_ALG_RSA)
        return TPM_RCS_KEY + RC_RSA_Encrypt_keyHandle;
    // selected key must have the decryption attribute
    if(!IS_ATTRIBUTE(rsaKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
        return TPM_RCS_ATTRIBUTES + RC_RSA_Encrypt_keyHandle;

    // Is there a label?
    if(!IsLabelProperlyFormatted(&in->label.b))
        return TPM_RCS_VALUE + RC_RSA_Encrypt_label;
    // Command Output
    // Select a scheme for encryption
    scheme = CryptRsaSelectScheme(in->keyHandle, &in->inScheme);
    if(scheme == NULL)
        return TPM_RCS_SCHEME + RC_RSA_Encrypt_inScheme;

    // Encryption.  TPM_RC_VALUE, or TPM_RC_SCHEME errors my be returned buy
    // CryptEncyptRSA.
    out->outData.t.size = sizeof(out->outData.t.buffer);

    result              = CryptRsaEncrypt(
        &out->outData, &in->message.b, rsaKey, scheme, &in->label.b, NULL);
    return result;
}

#endif  // CC_RSA_Encrypt
```

7.9 /tpm/src/command/Asymmetric/ZGen_2Phase.c

```
#include "Tpm.h"
#include "ZGen_2Phase_fp.h"

#if CC_ZGen_2Phase  // Conditional expansion of this file

// This command uses the TPM to recover one or two Z values in a two phase key
// exchange protocol
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               key referenced by 'keyA' is restricted or
//                                      not a decrypt key
//      TPM_RC_ECC_POINT                'inQsB' or  'inQeB' is not on the curve of
//                                      the key reference by 'keyA'
//      TPM_RC_KEY                      key referenced by 'keyA' is not an ECC key
//      TPM_RC_SCHEME                   the scheme of the key referenced by 'keyA'
//                                      is not TPM_ALG_NULL, TPM_ALG_ECDH,
//                                      TPM_ALG_ECMQV or TPM_ALG_SM2
TPM_RC
TPM2_ZGen_2Phase(ZGen_2Phase_In*  in,  // IN: input parameter list
                 ZGen_2Phase_Out* out  // OUT: output parameter list
)
{
    TPM_RC              result;
    OBJECT*             eccKey;
    TPM2B_ECC_PARAMETER r;
    TPM_ALG_ID          scheme;

    // Input Validation

    eccKey = HandleToObject(in->keyA);

    // keyA must be an ECC key
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RCS_KEY + RC_ZGen_2Phase_keyA;

    // keyA must not be restricted and must be a decrypt key
    if(IS_ATTRIBUTE(eccKey->publicArea.objectAttributes, TPMA_OBJECT, restricted)
       || !IS_ATTRIBUTE(eccKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
        return TPM_RCS_ATTRIBUTES + RC_ZGen_2Phase_keyA;

    // if the scheme of keyA is TPM_ALG_NULL, then use the input scheme; otherwise
    // the input scheme must be the same as the scheme of keyA
    scheme = eccKey->publicArea.parameters.asymDetail.scheme.scheme;
    if(scheme != TPM_ALG_NULL)
    {
        if(scheme != in->inScheme)
            return TPM_RCS_SCHEME + RC_ZGen_2Phase_inScheme;
    }
    else
        scheme = in->inScheme;
    if(scheme == TPM_ALG_NULL)
        return TPM_RCS_SCHEME + RC_ZGen_2Phase_inScheme;

    // Input points must be on the curve of keyA
    if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
                               &in->inQsB.point))
        return TPM_RCS_ECC_POINT + RC_ZGen_2Phase_inQsB;

    if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
                               &in->inQeB.point))
        return TPM_RCS_ECC_POINT + RC_ZGen_2Phase_inQeB;

    if(!CryptGenerateR(
           &r, &in->counter, eccKey->publicArea.parameters.eccDetail.curveID, NULL))
        return TPM_RCS_VALUE + RC_ZGen_2Phase_counter;
```

```
    // Command Output

    result =
        CryptEcc2PhaseKeyExchange(&out->outZ1.point,
                                  &out->outZ2.point,
                                  eccKey->publicArea.parameters.eccDetail.curveID,
                                  scheme,
                                  &eccKey->sensitive.sensitive.ecc,
                                  &r,
                                  &in->inQsB.point,
                                  &in->inQeB.point);
    if(result == TPM_RC_SCHEME)
        return TPM_RCS_SCHEME + RC_ZGen_2Phase_inScheme;

    if(result == TPM_RC_SUCCESS)
        CryptEndCommit(in->counter);

    return result;
}
#endif  // CC_ZGen_2Phase
```

7.10 /tpm/src/command/AttachedComponent/AC_GetCapability.c

```
#include "Tpm.h"
#include "AC_GetCapability_fp.h"
#include "AC_spt_fp.h"

#if CC_AC_GetCapability  // Conditional expansion of this file

/*(See part 3 specification)
// This command returns various information regarding Attached Components
*/
TPM_RC
TPM2_AC_GetCapability(AC_GetCapability_In*  in,  // IN: input parameter list
                      AC_GetCapability_Out* out  // OUT: output parameter list
)
{
    // Command Output
    out->moreData =
        AcCapabilitiesGet(in->ac, in->capability, in->count, &out->capabilitiesData);

    return TPM_RC_SUCCESS;
}

#endif  // CC_AC_GetCapability
```

7.11 /tpm/src/command/AttachedComponent/AC_Send.c

```
#include "Tpm.h"
#include "AC_Send_fp.h"
#include "AC_spt_fp.h"

#if CC_AC_Send  // Conditional expansion of this file

/*(See part 3 specification)
// Duplicate a loaded object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   key to duplicate has 'fixedParent' SET
//      TPM_RC_HASH         for an RSA key, the nameAlg digest size for the
//                          newParent is not compatible with the key size
//      TPM_RC_HIERARCHY    'encryptedDuplication' is SET and 'newParentHandle'
//                          specifies Null Hierarchy
```

```
//      TPM_RC_KEY          'newParentHandle' references invalid ECC key (public
//                          point not on the curve)
//      TPM_RC_SIZE         input encryption key size does not match the
//                          size specified in symmetric algorithm
//      TPM_RC_SYMMETRIC    'encryptedDuplication' is SET but no symmetric
//                          algorithm is provided
//      TPM_RC_TYPE         'newParentHandle' is neither a storage key nor
//                          TPM_RH_NULL; or the object has a NULL nameAlg
//      TPM_RC_VALUE        for an RSA newParent, the sizes of the digest and
//                          the encryption key are too large to be OAEP encoded
TPM_RC
TPM2_AC_Send(AC_Send_In*  in,  // IN: input parameter list
             AC_Send_Out* out  // OUT: output parameter list
)
{
    NV_REF     locator;
    TPM_HANDLE nvAlias = ((in->ac - AC_FIRST) + NV_AC_FIRST);
    NV_INDEX*  nvIndex = NvGetIndexInfo(nvAlias, &locator);
    OBJECT*    object  = HandleToObject(in->sendObject);
    TPM_RC     result;
    // Input validation
    // If there is an NV alias, then the index must allow the authorization provided
    if(nvIndex != NULL)
    {
        // Common access checks, NvWriteAccessCheck() may return
        // TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
        result = NvWriteAccessChecks(
            in->authHandle, nvAlias, nvIndex->publicArea.attributes);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    // If 'ac' did not have an alias then the authorization had to be with either
    // platform or owner authorization. The type of TPMI_RH_NV_AUTH only allows
    // owner or platform or an NV index. If it was a valid index, it would have had
    // an alias and be processed above, so only success here is if this is a
    // permanent handle.
    else if(HandleGetType(in->authHandle) != TPM_HT_PERMANENT)
        return TPM_RCS_HANDLE + RC_AC_Send_authHandle;
    // Make sure that the object to be duplicated has the right attributes
    if(IS_ATTRIBUTE(
           object->publicArea.objectAttributes, TPMA_OBJECT, encryptedDuplication)
       || IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, fixedParent)
       || IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, fixedTPM))
        return TPM_RCS_ATTRIBUTES + RC_AC_Send_sendObject;
    // Command output
    // Do the implementation dependent send
    return AcSendObject(in->ac, object, &out->acDataOut);
}

#endif  // TPM_CC_AC_Send
```

7.12 /tpm/src/command/AttachedComponent/AC_spt.c

```
//** Introduction
// This code in this clause is provided for testing of the TPM's command interface.
// The implementation of Attached Components is not expected to be as shown in this
// code.

//** Includes
#include "Tpm.h"
#include "AC_spt_fp.h"

// This is the simulated AC data. This should be present in an actual implementation.
#if 1
```

```
typedef struct
{
    TPMI_RH_AC            ac;
    TPML_AC_CAPABILITIES* acData;

} acCapabilities;

TPML_AC_CAPABILITIES acData0001 = {1, {{TPM_AT_PV1, 0x01234567}}};

acCapabilities       ac[1]      = {{0x0001, &acData0001}};

#  define NUM_AC (sizeof(ac) / sizeof(acCapabilities))

#endif  // 1 The simulated AC data

//** Functions

//*** AcToCapabilities()
// This function returns a pointer to a list of AC capabilities.
TPML_AC_CAPABILITIES* AcToCapabilities(TPMI_RH_AC component  // IN: component
)
{
    UINT32 index;
    //
    for(index = 0; index < NUM_AC; index++)
    {
        if(ac[index].ac == component)
            return ac[index].acData;
    }
    return NULL;
}

//*** AcIsAccessible()
// Function to determine if an AC handle references an actual AC
//  Return Type: BOOL
BOOL AcIsAccessible(TPM_HANDLE acHandle)
{
    // In this implementation, the AC exists if there are some capabilities to go
    // with the handle
    return AcToCapabilities(acHandle) != NULL;
}

//*** AcCapabilitiesGet()
// This function returns a list of capabilities associated with an AC
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
AcCapabilitiesGet(TPMI_RH_AC            component,      // IN: the component
                  TPM_AT                type,           // IN: start capability type
                  UINT32                count,          // IN: requested number
                  TPML_AC_CAPABILITIES* capabilityList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;
    // Get the list of capabilities and their values associated with the AC
    TPML_AC_CAPABILITIES* capabilities;

    pAssert(HandleGetType(component) == TPM_HT_AC);
    capabilities = AcToCapabilities(component);

    // Initialize output handle list
    capabilityList->count = 0;
    if(count > MAX_AC_CAPABILITIES)
        count = MAX_AC_CAPABILITIES;
```

```
    if(capabilities != NULL)
    {
        // Find the first capability less than or equal to type
        for(i = 0; i < capabilities->count; i++)
        {
            if(capabilities->acCapabilities[i].tag >= type)
            {
                // copy the capabilities until we run out or fill the list
                for(; (capabilityList->count < count) && (i < capabilities->count);
                    i++)
                {
                    capabilityList->acCapabilities[capabilityList->count] =
                        capabilities->acCapabilities[i];
                    capabilityList->count++;
                }
                more = i < capabilities->count;
            }
        }
    }
    return more;
}

//*** AcSendObject()
// Stub to handle sending of an AC object
//  Return Type: TPM_RC
TPM_RC
AcSendObject(TPM_HANDLE      acHandle,  // IN: Handle of AC receiving object
             OBJECT*         object,    // IN: object structure to send
             TPMS_AC_OUTPUT* acDataOut  // OUT: results of operation
)
{
    NOT_REFERENCED(object);
    NOT_REFERENCED(acHandle);
    acDataOut->tag = TPM_AT_ERROR;  // indicate that the response contains an
                                    // error code
    acDataOut->data = TPM_AE_NONE;  // but there is no error.

    return TPM_RC_SUCCESS;
}
```

7.13 /tpm/src/command/AttachedComponent/Policy_AC_SendSelect.c

```
#include "Tpm.h"
#include "Policy_AC_SendSelect_fp.h"

#if CC_Policy_AC_SendSelect  // Conditional expansion of this file

/*(See part 3 specification)
// allows qualification of attached component and object to be sent.
*/
//  Return Type: TPM_RC
//      TPM_RC_COMMAND_CODE   'commandCode' of 'policySession' is not empty
//      TPM_RC_CPHASH         'cpHash' of 'policySession' is not empty
TPM_RC
TPM2_Policy_AC_SendSelect(Policy_AC_SendSelect_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    HASH_STATE hashState;
    TPM_CC     commandCode = TPM_CC_Policy_AC_SendSelect;

    // Input Validation

    // Get pointer to the session structure
```

```
    session = SessionGet(in->policySession);

    // cpHash in session context must be empty
    if(session->u1.cpHash.t.size != 0)
        return TPM_RC_CPHASH;
    // commandCode in session context must be empty
    if(session->commandCode != 0)
        return TPM_RC_COMMAND_CODE;
    // Internal Data Update
    // Update name hash
    session->u1.cpHash.t.size = CryptHashStart(&hashState, session->authHashAlg);

    //  add objectName
    CryptDigestUpdate2B(&hashState, &in->objectName.b);

    // add authHandleName
    CryptDigestUpdate2B(&hashState, &in->authHandleName.b);

    //  add ac name
    CryptDigestUpdate2B(&hashState, &in->acName.b);

    //  complete hash
    CryptHashEnd2B(&hashState, &session->u1.cpHash.b);

    // update policy hash
    // Old policyDigest size should be the same as the new policyDigest size since
    // they are using the same hash algorithm
    session->u2.policyDigest.t.size =
        CryptHashStart(&hashState, session->authHashAlg);
    //  add old policy
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add command code
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add objectName
    if(in->includeObject == YES)
        CryptDigestUpdate2B(&hashState, &in->objectName.b);

    //  add authHandleName
    CryptDigestUpdate2B(&hashState, &in->authHandleName.b);

    // add acName
    CryptDigestUpdate2B(&hashState, &in->acName.b);

    //  add includeObject
    CryptDigestUpdateInt(&hashState, sizeof(TPMI_YES_NO), in->includeObject);

    //  complete digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // set commandCode in session context
    session->commandCode = TPM_CC_AC_Send;

    return TPM_RC_SUCCESS;
}

#endif  // CC_Policy_AC_SendSelect
```

7.14 /tpm/src/command/Attestation/Attest_spt.c

```
//** Includes
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "Marshal.h"
```

```
//** Functions

//***FillInAttestInfo()
// Fill in common fields of TPMS_ATTEST structure.
void FillInAttestInfo(
    TPMI_DH_OBJECT   signHandle,  // IN: handle of signing object
    TPMT_SIG_SCHEME* scheme,      // IN/OUT: scheme to be used for signing
    TPM2B_DATA*      data,        // IN: qualifying data
    TPMS_ATTEST*     attest       // OUT: attest structure
)
{
    OBJECT* signObject = HandleToObject(signHandle);

    // Magic number
    attest->magic = TPM_GENERATED_VALUE;

    if(signObject == NULL)
    {
        // The name for a null handle is TPM_RH_NULL
        // This is defined because UINT32_TO_BYTE_ARRAY does a cast. If the
        // size of the cast is smaller than a constant, the compiler warns
        // about the truncation of a constant value.
        TPM_HANDLE nullHandle          = TPM_RH_NULL;
        attest->qualifiedSigner.t.size = sizeof(TPM_HANDLE);
        UINT32_TO_BYTE_ARRAY(nullHandle, attest->qualifiedSigner.t.name);
    }
    else
    {
        // Certifying object qualified name
        // if the scheme is anonymous, this is an empty buffer
        if(CryptIsSchemeAnonymous(scheme->scheme))
            attest->qualifiedSigner.t.size = 0;
        else
            attest->qualifiedSigner = signObject->qualifiedName;
    }
    // current clock in plain text
    TimeFillInfo(&attest->clockInfo);

    // Firmware version in plain text
    attest->firmwareVersion = ((UINT64)gp.firmwareV1 << (sizeof(UINT32) * 8));
    attest->firmwareVersion += gp.firmwareV2;

    // Check the hierarchy of sign object.  For NULL sign handle, the hierarchy
    // will be TPM_RH_NULL
    if((signObject == NULL)
       || (!signObject->attributes.epsHierarchy
           && !signObject->attributes.ppsHierarchy))
    {
        // For signing key that is not in platform or endorsement hierarchy,
        // obfuscate the reset, restart and firmware version information
        UINT64 obfuscation[2];
        CryptKDFa(CONTEXT_INTEGRITY_HASH_ALG,
                  &gp.shProof.b,
                  OBFUSCATE_STRING,
                  &attest->qualifiedSigner.b,
                  NULL,
                  128,
                  (BYTE*)&obfuscation[0],
                  NULL,
                  FALSE);
        // Obfuscate data
        attest->firmwareVersion += obfuscation[0];
        attest->clockInfo.resetCount += (UINT32)(obfuscation[1] >> 32);
        attest->clockInfo.restartCount += (UINT32)obfuscation[1];
    }
```

```
    // External data
    if(CryptIsSchemeAnonymous(scheme->scheme))
        attest->extraData.t.size = 0;
    else
    {
        // If we move the data to the attestation structure, then it is not
        // used in the signing operation except as part of the signed data
        attest->extraData = *data;
        data->t.size      = 0;
    }
}

//***SignAttestInfo()
// Sign a TPMS_ATTEST structure. If signHandle is TPM_RH_NULL, a null signature
// is returned.
//
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   'signHandle' references not a signing key
//      TPM_RC_SCHEME       'scheme' is not compatible with 'signHandle' type
//      TPM_RC_VALUE        digest generated for the given 'scheme' is greater than
//                          the modulus of 'signHandle' (for an RSA key);
//                          invalid commit status or failed to generate "r" value
//                          (for an ECC key)
TPM_RC
SignAttestInfo(OBJECT*           signKey,          // IN: sign object
               TPMT_SIG_SCHEME* scheme,           // IN: sign scheme
               TPMS_ATTEST*     certifyInfo,      // IN: the data to be signed
               TPM2B_DATA*      qualifyingData,   // IN: extra data for the signing
                                                  //     process
               TPM2B_ATTEST* attest,              // OUT: marshaled attest blob to be
                                                  //     signed
               TPMT_SIGNATURE* signature          // OUT: signature
)
{
    BYTE*        buffer;
    HASH_STATE   hashState;
    TPM2B_DIGEST digest;
    TPM_RC       result;

    // Marshal TPMS_ATTEST structure for hash
    buffer         = attest->t.attestationData;
    attest->t.size = TPMS_ATTEST_Marshal(certifyInfo, &buffer, NULL);

    if(signKey == NULL)
    {
        signature->sigAlg = TPM_ALG_NULL;
        result            = TPM_RC_SUCCESS;
    }
    else
    {
        TPMI_ALG_HASH hashAlg;
        // Compute hash
        hashAlg = scheme->details.any.hashAlg;
        // need to set the receive buffer to get something put in it
        digest.t.size = sizeof(digest.t.buffer);
        digest.t.size = CryptHashBlock(hashAlg,
                                       attest->t.size,
                                       attest->t.attestationData,
                                       digest.t.size,
                                       digest.t.buffer);
        // If there is qualifying data, need to rehash the data
        // hash(qualifyingData || hash(attestationData))
        if(qualifyingData->t.size != 0)
        {
            CryptHashStart(&hashState, hashAlg);
            CryptDigestUpdate2B(&hashState, &qualifyingData->b);
```

```
            CryptDigestUpdate2B(&hashState, &digest.b);
            CryptHashEnd2B(&hashState, &digest.b);
        }
        // Sign the hash. A TPM_RC_VALUE, TPM_RC_SCHEME, or
        // TPM_RC_ATTRIBUTES error may be returned at this point
        result = CryptSign(signKey, scheme, &digest, signature);

        // Since the clock is used in an attestation, the state in NV is no longer
        // "orderly" with respect to the data in RAM if the signature is valid
        if(result == TPM_RC_SUCCESS)
        {
            // Command uses the clock so need to clear the orderly state if it is
            // set.
            result = NvClearOrderly();
        }
    }
    return result;
}

//*** IsSigningObject()
// Checks to see if the object is OK for signing. This is here rather than in
// Object_spt.c because all the attestation commands use this file but not
// Object_spt.c.
//  Return Type: BOOL
//      TRUE(1)         object may sign
//      FALSE(0)        object may not sign
BOOL IsSigningObject(OBJECT* object  // IN:
)
{
    return ((object == NULL)
            || ((IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, sign)
                 && object->publicArea.type != TPM_ALG_SYMCIPHER)));
}
```

7.15 /tpm/src/command/Attestation/Certify.c

```
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "Certify_fp.h"

#if CC_Certify  // Conditional expansion of this file

/*(See part 3 specification)
// prove an object with a specific Name is loaded in the TPM
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          key referenced by 'signHandle' is not a signing key
//      TPM_RC_SCHEME       'inScheme' is not compatible with 'signHandle'
//      TPM_RC_VALUE        digest generated for 'inScheme' is greater or has larger
//                          size than the modulus of 'signHandle', or the buffer for
//                          the result in 'signature' is too small (for an RSA key);
//                          invalid commit status (for an ECC key with a split scheme)
TPM_RC
TPM2_Certify(Certify_In*  in,  // IN: input parameter list
             Certify_Out* out  // OUT: output parameter list
)
{
    TPMS_ATTEST certifyInfo;
    OBJECT*     signObject      = HandleToObject(in->signHandle);
    OBJECT*     certifiedObject = HandleToObject(in->objectHandle);
    // Input validation
    if(!IsSigningObject(signObject))
        return TPM_RCS_KEY + RC_Certify_signHandle;
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_Certify_inScheme;
```

```
    // Command Output
    // Filling in attest information
    // Common fields
    FillInAttestInfo(
        in->signHandle, &in->inScheme, &in->qualifyingData, &certifyInfo);

    // Certify specific fields
    certifyInfo.type = TPM_ST_ATTEST_CERTIFY;
    // NOTE: the certified object is not allowed to be TPM_ALG_NULL so
    // 'certifiedObject' will never be NULL
    certifyInfo.attested.certify.name = certifiedObject->name;

    // When using an anonymous signing scheme, need to set the qualified Name to the
    // empty buffer to avoid correlation between keys
    if(CryptIsSchemeAnonymous(in->inScheme.scheme))
        certifyInfo.attested.certify.qualifiedName.t.size = 0;
    else
        certifyInfo.attested.certify.qualifiedName = certifiedObject->qualifiedName;

    // Sign attestation structure.  A NULL signature will be returned if
    // signHandle is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned
    // by SignAttestInfo()
    return SignAttestInfo(signObject,
                          &in->inScheme,
                          &certifyInfo,
                          &in->qualifyingData,
                          &out->certifyInfo,
                          &out->signature);
}

#endif  // CC_Certify
```

7.16 /tpm/src/command/Attestation/CertifyCreation.c

```
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "CertifyCreation_fp.h"

#if CC_CertifyCreation  // Conditional expansion of this file

/*(See part 3 specification)
// Prove the association between an object and its creation data
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          key referenced by 'signHandle' is not a signing key
//      TPM_RC_SCHEME       'inScheme' is not compatible with 'signHandle'
//      TPM_RC_TICKET       'creationTicket' does not match 'objectHandle'
//      TPM_RC_VALUE        digest generated for 'inScheme' is greater or has larger
//                          size than the modulus of 'signHandle', or the buffer for
//                          the result in 'signature' is too small (for an RSA key);
//                          invalid commit status (for an ECC key with a split
scheme).
TPM_RC
TPM2_CertifyCreation(CertifyCreation_In*  in,  // IN: input parameter list
                     CertifyCreation_Out* out  // OUT: output parameter list
)
{
    TPM_RC            result = TPM_RC_SUCCESS;
    TPMT_TK_CREATION ticket;
    TPMS_ATTEST       certifyInfo;
    OBJECT*           certified  = HandleToObject(in->objectHandle);
    OBJECT*           signObject = HandleToObject(in->signHandle);
    // Input Validation
```

```
    if(!IsSigningObject(signObject))
        return TPM_RCS_KEY + RC_CertifyCreation_signHandle;
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_CertifyCreation_inScheme;

    // CertifyCreation specific input validation
    // Re-compute ticket
    result = TicketComputeCreation(
        in->creationTicket.hierarchy, &certified->name, &in->creationHash, &ticket);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Compare ticket
    if(!MemoryEqual2B(&ticket.digest.b, &in->creationTicket.digest.b))
        return TPM_RCS_TICKET + RC_CertifyCreation_creationTicket;

    // Command Output
    // Common fields
    FillInAttestInfo(
        in->signHandle, &in->inScheme, &in->qualifyingData, &certifyInfo);

    // CertifyCreation specific fields
    // Attestation type
    certifyInfo.type                          = TPM_ST_ATTEST_CREATION;
    certifyInfo.attested.creation.objectName = certified->name;

    // Copy the creationHash
    certifyInfo.attested.creation.creationHash = in->creationHash;

    // Sign attestation structure.  A NULL signature will be returned if
    // signObject is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
    // this point
    return SignAttestInfo(signObject,
                          &in->inScheme,
                          &certifyInfo,
                          &in->qualifyingData,
                          &out->certifyInfo,
                          &out->signature);
}

#endif  // CC_CertifyCreation
```

7.17 /tpm/src/command/Attestation/CertifyX509.c

```
#include "Tpm.h"
#include "CertifyX509_fp.h"
#include "X509.h"
#include "TpmASN1_fp.h"
#include "X509_spt_fp.h"
#include "Attest_spt_fp.h"
#if CERTIFYX509_DEBUG
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#  include <platform_interface/tpm_to_platform_interface.h>
#endif

#if CC_CertifyX509  // Conditional expansion of this file

/*(See part 3 specification)
// Certify using an X509-formatted certificate
*/
// return type: TPM_RC
//      TPM_RC_ATTRIBUTES       the attributes of 'objectHandle' are not compatible
//                              with the KeyUsage or TPMA_OBJECT values in the
```

```
//                              extensions fields
//      TPM_RC_BINDING          the public and private portions of the key are not
//                              properly bound.
//      TPM_RC_HASH             the hash algorithm in the scheme is not supported
//      TPM_RC_KEY              'signHandle' does not reference a signing key;
//      TPM_RC_SCHEME           the scheme is not compatible with sign key type,
//                              or input scheme is not compatible with default
//                              scheme, or the chosen scheme is not a valid
//                              sign scheme
//      TPM_RC_VALUE            most likely a problem with the format of
//                              'partialCertificate'
TPM_RC
TPM2_CertifyX509(CertifyX509_In*  in,  // IN: input parameter list
                 CertifyX509_Out* out  // OUT: output parameter list
)
{
    TPM_RC               result;
    OBJECT*              signKey = HandleToObject(in->signHandle);
    OBJECT*              object  = HandleToObject(in->objectHandle);
    HASH_STATE           hash;
    INT16                length;  // length for a tagged element
    ASN1UnmarshalContext ctx;
    ASN1MarshalContext   ctxOut;
    // certTBS holds an array of pointers and lengths. Each entry references the
    // corresponding value in a TBSCertificate structure. For example, the 1th
    // element references the version number
    stringRef certTBS[REF_COUNT] = {{0}};
#   define ALLOWED_SEQUENCES (SUBJECT_PUBLIC_KEY_REF - SIGNATURE_REF)
    stringRef partial[ALLOWED_SEQUENCES] = {{0}};
    INT16     countOfSequences           = 0;
    INT16     i;
    //
#   if CERTIFYX509_DEBUG
    DebugFileInit();
    DebugDumpBuffer(in->partialCertificate.t.size,
                    in->partialCertificate.t.buffer,
                    "partialCertificate");
#   endif

    // Input Validation
    if(in->reserved.b.size != 0)
        return TPM_RC_SIZE + RC_CertifyX509_reserved;
    // signing key must be able to sign
    if(!IsSigningObject(signKey))
        return TPM_RCS_KEY + RC_CertifyX509_signHandle;
    // Pick a scheme for sign.  If the input sign scheme is not compatible with
    // the default scheme, return an error.
    if(!CryptSelectSignScheme(signKey, &in->inScheme))
        return TPM_RCS_SCHEME + RC_CertifyX509_inScheme;
    // Make sure that the public Key encoding is known
    if(X509AddPublicKey(NULL, object) == 0)
        return TPM_RCS_ASYMMETRIC + RC_CertifyX509_objectHandle;
    // Unbundle 'partialCertificate'.
    // Initialize the unmarshaling context
    if(!ASN1UnmarshalContextInitialize(
           &ctx, in->partialCertificate.t.size, in->partialCertificate.t.buffer))
        return TPM_RCS_VALUE + RC_CertifyX509_partialCertificate;
    // Make sure that this is a constructed SEQUENCE
    length = ASN1NextTag(&ctx);
    // Must be a constructed SEQUENCE that uses all of the input parameter
    if((ctx.tag != (ASN1_CONSTRUCTED_SEQUENCE))
       || ((ctx.offset + length) != in->partialCertificate.t.size))
        return TPM_RCS_SIZE + RC_CertifyX509_partialCertificate;

    // This scans through the contents of the outermost SEQUENCE. This would be the
    // 'issuer', 'validity', 'subject', 'issuerUniqueID' (optional),
```

```
    // 'subjectUniqueID' (optional), and 'extensions.'
    while(ctx.offset < ctx.size)
    {
        INT16 startOfElement = ctx.offset;
        //
        // Read the next tag and length field.
        length = ASN1NextTag(&ctx);
        if(length < 0)
            break;
        if(ctx.tag == ASN1_CONSTRUCTED_SEQUENCE)
        {
            if(countOfSequences < ALLOWED_SEQUENCES)
            {
                partial[countOfSequences].buf = &ctx.buffer[startOfElement];
                ctx.offset += length;
                partial[countOfSequences].len = (INT16)ctx.offset - startOfElement;
            }
            countOfSequences++;
            if(countOfSequences > ALLOWED_SEQUENCES)
                break;
        }
        else if(ctx.tag == X509_EXTENSIONS)
        {
            if(certTBS[EXTENSIONS_REF].len != 0)
                return TPM_RCS_VALUE + RC_CertifyX509_partialCertificate;
            certTBS[EXTENSIONS_REF].buf = &ctx.buffer[startOfElement];
            ctx.offset += length;
            certTBS[EXTENSIONS_REF].len = (INT16)ctx.offset - startOfElement;
        }
        else
            return TPM_RCS_VALUE + RC_CertifyX509_partialCertificate;
    }
    // Make sure that we used all of the data and found at least the required
    // number of elements.
    if((ctx.offset != ctx.size) || (countOfSequences < 3) || (countOfSequences > 4)
       || (certTBS[EXTENSIONS_REF].buf == NULL))
        return TPM_RCS_VALUE + RC_CertifyX509_partialCertificate;
    // Now that we know how many sequences there were, we can put them where they
    // belong
    for(i = 0; i < countOfSequences; i++)
        certTBS[SUBJECT_KEY_REF - i] = partial[countOfSequences - 1 - i];

    // If only three SEQUENCES, then the TPM needs to produce the signature algorithm.
    // See if it can
    if((countOfSequences == 3)
       && (X509AddSigningAlgorithm(NULL, signKey, &in->inScheme) == 0))
        return TPM_RCS_SCHEME + RC_CertifyX509_signHandle;

    // Process the extensions
    result = X509ProcessExtensions(object, &certTBS[EXTENSIONS_REF]);
    if(result != TPM_RC_SUCCESS)
        // If the extension has the TPMA_OBJECT extension and the attributes don't
        // match, then the error code will be TPM_RCS_ATTRIBUTES. Otherwise, the error
        // indicates a malformed partialCertificate.
        return result
               + ((result == TPM_RCS_ATTRIBUTES) ? RC_CertifyX509_objectHandle
                                                 : RC_CertifyX509_partialCertificate);
    // Command Output
    // Create the addedToCertificate values

    // Build the addedToCertificate from the bottom up.
    // Initialize the context structure
    ASN1InitializeMarshalContext(&ctxOut,
                                 sizeof(out->addedToCertificate.t.buffer),
                                 out->addedToCertificate.t.buffer);
    // Place a marker for the overall context
```

```
    ASN1StartMarshalContext(&ctxOut);  // SEQUENCE for addedToCertificate

    // Add the subject public key descriptor
    certTBS[SUBJECT_PUBLIC_KEY_REF].len = X509AddPublicKey(&ctxOut, object);
    certTBS[SUBJECT_PUBLIC_KEY_REF].buf = ctxOut.buffer + ctxOut.offset;
    // If the caller didn't provide the algorithm identifier, create it
    if(certTBS[SIGNATURE_REF].len == 0)
    {
        certTBS[SIGNATURE_REF].len =
            X509AddSigningAlgorithm(&ctxOut, signKey, &in->inScheme);
        certTBS[SIGNATURE_REF].buf = ctxOut.buffer + ctxOut.offset;
    }
    // Create the serial number value. Use the out->tbsDigest as scratch.
    {
        TPM2B* digest = &out->tbsDigest.b;
        //
        digest->size = (INT16)CryptHashStart(&hash, signKey->publicArea.nameAlg);
        pAssert(digest->size != 0);

        // The serial number size is the smaller of the digest and the vendor-defined
        // value
        digest->size = MIN(digest->size, SIZE_OF_X509_SERIAL_NUMBER);
        // Add all the parts of the certificate other than the serial number
        // and version number
        for(i = SIGNATURE_REF; i < REF_COUNT; i++)
            CryptDigestUpdate(&hash, certTBS[i].len, certTBS[i].buf);
        // throw in the Name of the signing key...
        CryptDigestUpdate2B(&hash, &signKey->name.b);
        // ...and the Name of the signed key.
        CryptDigestUpdate2B(&hash, &object->name.b);
        // Done
        CryptHashEnd2B(&hash, digest);
    }

    // Add the serial number
    certTBS[SERIAL_NUMBER_REF].len =
        ASN1PushInteger(&ctxOut, out->tbsDigest.t.size, out->tbsDigest.t.buffer);
    certTBS[SERIAL_NUMBER_REF].buf = ctxOut.buffer + ctxOut.offset;

    // Add the static version number
    ASN1StartMarshalContext(&ctxOut);
    ASN1PushUINT(&ctxOut, 2);
    certTBS[VERSION_REF].len =
        ASN1EndEncapsulation(&ctxOut, ASN1_APPLICAIION_SPECIFIC);
    certTBS[VERSION_REF].buf = ctxOut.buffer + ctxOut.offset;

    // Create a fake tag and length for the TBS in the space used for
    // 'addedToCertificate'
    {
        for(length = 0, i = 0; i < REF_COUNT; i++)
            length += certTBS[i].len;
        // Put a fake tag and length into the buffer for use in the tbsDigest
        certTBS[ENCODED_SIZE_REF].len =
            ASN1PushTagAndLength(&ctxOut, ASN1_CONSTRUCTED_SEQUENCE, length);
        certTBS[ENCODED_SIZE_REF].buf = ctxOut.buffer + ctxOut.offset;
        // Restore the buffer pointer to add back the number of octets used for the
        // tag and length
        ctxOut.offset += certTBS[ENCODED_SIZE_REF].len;
    }
    // sanity check
    if(ctxOut.offset < 0)
        return TPM_RC_FAILURE;
    // Create the tbsDigest to sign
    out->tbsDigest.t.size = CryptHashStart(&hash, in->inScheme.details.any.hashAlg);
    for(i = 0; i < REF_COUNT; i++)
        CryptDigestUpdate(&hash, certTBS[i].len, certTBS[i].buf);
```

```
    CryptHashEnd2B(&hash, &out->tbsDigest.b);

#  if CERTIFYX509_DEBUG
    {
        BYTE  fullTBS[4096];
        BYTE* fill = fullTBS;
        int   j;
        for(j = 0; j < REF_COUNT; j++)
        {
            MemoryCopy(fill, certTBS[j].buf, certTBS[j].len);
            fill += certTBS[j].len;
        }
        DebugDumpBuffer((int)(fill - &fullTBS[0]), fullTBS, "\nfull TBS");
    }
#  endif

    // Finish up the processing of addedToCertificate
    // Create the actual tag and length for the addedToCertificate structure
    out->addedToCertificate.t.size =
        ASN1EndEncapsulation(&ctxOut, ASN1_CONSTRUCTED_SEQUENCE);
    // Now move all the addedToContext to the start of the buffer
    MemoryCopy(out->addedToCertificate.t.buffer,
               ctxOut.buffer + ctxOut.offset,
               out->addedToCertificate.t.size);
#  if CERTIFYX509_DEBUG
    DebugDumpBuffer(out->addedToCertificate.t.size,
                    out->addedToCertificate.t.buffer,
                    "\naddedToCertificate");
#  endif
    // only thing missing is the signature
    result = CryptSign(signKey, &in->inScheme, &out->tbsDigest, &out->signature);

    return result;
}

#endif  // CC_CertifyX509
```

7.18 /tpm/src/command/Attestation/GetCommandAuditDigest.c

```
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "GetCommandAuditDigest_fp.h"

#if CC_GetCommandAuditDigest  // Conditional expansion of this file

/*(See part 3 specification)
// Get current value of command audit log
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          key referenced by 'signHandle' is not a signing key
//      TPM_RC_SCHEME       'inScheme' is incompatible with 'signHandle' type; or
//                          both 'scheme' and key's default scheme are empty; or
//                          'scheme' is empty while key's default scheme requires
//                          explicit input scheme (split signing); or
//                          non-empty default key scheme differs from 'scheme'
//      TPM_RC_VALUE        digest generated for the given 'scheme' is greater than
//                          the modulus of 'signHandle' (for an RSA key);
//                          invalid commit status or failed to generate "r" value
//                          (for an ECC key)
TPM_RC
TPM2_GetCommandAuditDigest(
    GetCommandAuditDigest_In*  in,  // IN: input parameter list
    GetCommandAuditDigest_Out* out  // OUT: output parameter list
)
{
```

```
    TPM_RC      result;
    TPMS_ATTEST auditInfo;
    OBJECT*     signObject = HandleToObject(in->signHandle);
    // Input validation
    if(!IsSigningObject(signObject))
        return TPM_RCS_KEY + RC_GetCommandAuditDigest_signHandle;
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_GetCommandAuditDigest_inScheme;

    // Command Output
    // Fill in attest information common fields
    FillInAttestInfo(in->signHandle, &in->inScheme, &in->qualifyingData, &auditInfo);

    // CommandAuditDigest specific fields
    auditInfo.type                                = TPM_ST_ATTEST_COMMAND_AUDIT;
    auditInfo.attested.commandAudit.digestAlg     = gp.auditHashAlg;
    auditInfo.attested.commandAudit.auditCounter = gp.auditCounter;

    // Copy command audit log
    auditInfo.attested.commandAudit.auditDigest = gr.commandAuditDigest;
    CommandAuditGetDigest(&auditInfo.attested.commandAudit.commandDigest);

    // Sign attestation structure.  A NULL signature will be returned if
    // signHandle is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
    // this point
    result = SignAttestInfo(signObject,
                            &in->inScheme,
                            &auditInfo,
                            &in->qualifyingData,
                            &out->auditInfo,
                            &out->signature);
    // Internal Data Update
    if(result == TPM_RC_SUCCESS && in->signHandle != TPM_RH_NULL)
        // Reset log
        gr.commandAuditDigest.t.size = 0;

    return result;
}

#endif  // CC_GetCommandAuditDigest
```

7.19 /tpm/src/command/Attestation/GetSessionAuditDigest.c

```
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "GetSessionAuditDigest_fp.h"

#if CC_GetSessionAuditDigest  // Conditional expansion of this file

/*(See part 3 specification)
// Get audit session digest
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          key referenced by 'signHandle' is not a signing key
//      TPM_RC_SCHEME       'inScheme' is incompatible with 'signHandle' type; or
//                          both 'scheme' and key's default scheme are empty; or
//                          'scheme' is empty while key's default scheme requires
//                          explicit input scheme (split signing); or
//                          non-empty default key scheme differs from 'scheme'
//      TPM_RC_TYPE         'sessionHandle' does not reference an audit session
//      TPM_RC_VALUE        digest generated for the given 'scheme' is greater than
//                          the modulus of 'signHandle' (for an RSA key);
//                          invalid commit status or failed to generate "r" value
//                          (for an ECC key)
```

```
TPM_RC
TPM2_GetSessionAuditDigest(
    GetSessionAuditDigest_In*  in,  // IN: input parameter list
    GetSessionAuditDigest_Out* out  // OUT: output parameter list
)
{
    SESSION*    session = SessionGet(in->sessionHandle);
    TPMS_ATTEST auditInfo;
    OBJECT*     signObject = HandleToObject(in->signHandle);
    // Input Validation
    if(!IsSigningObject(signObject))
        return TPM_RCS_KEY + RC_GetSessionAuditDigest_signHandle;
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_GetSessionAuditDigest_inScheme;

    // session must be an audit session
    if(session->attributes.isAudit == CLEAR)
        return TPM_RCS_TYPE + RC_GetSessionAuditDigest_sessionHandle;

    // Command Output
    // Fill in attest information common fields
    FillInAttestInfo(in->signHandle, &in->inScheme, &in->qualifyingData, &auditInfo);

    // SessionAuditDigest specific fields
    auditInfo.type                                 = TPM_ST_ATTEST_SESSION_AUDIT;
    auditInfo.attested.sessionAudit.sessionDigest = session->u2.auditDigest;

    // Exclusive audit session
    auditInfo.attested.sessionAudit.exclusiveSession =
        (g_exclusiveAuditSession == in->sessionHandle);

    // Sign attestation structure.  A NULL signature will be returned if
    // signObject is NULL.
    return SignAttestInfo(signObject,
                          &in->inScheme,
                          &auditInfo,
                          &in->qualifyingData,
                          &out->auditInfo,
                          &out->signature);
}

#endif  // CC_GetSessionAuditDigest
```

7.20 /tpm/src/command/Attestation/GetTime.c

```
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "GetTime_fp.h"

#if CC_GetTime  // Conditional expansion of this file

/*(See part 3 specification)
// Applies a time stamp to the passed blob (qualifyingData).
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          key referenced by 'signHandle' is not a signing key
//      TPM_RC_SCHEME       'inScheme' is incompatible with 'signHandle' type; or
//                          both 'scheme' and key's default scheme are empty; or
//                          'scheme' is empty while key's default scheme requires
//                          explicit input scheme (split signing); or
//                          non-empty default key scheme differs from 'scheme'
//      TPM_RC_VALUE        digest generated for the given 'scheme' is greater than
//                          the modulus of 'signHandle' (for an RSA key);
//                          invalid commit status or failed to generate "r" value
//                          (for an ECC key)
```

```
TPM_RC
TPM2_GetTime(GetTime_In*  in,  // IN: input parameter list
             GetTime_Out* out  // OUT: output parameter list
)
{
    TPMS_ATTEST timeInfo;
    OBJECT*     signObject = HandleToObject(in->signHandle);
    // Input Validation
    if(!IsSigningObject(signObject))
        return TPM_RCS_KEY + RC_GetTime_signHandle;
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_GetTime_inScheme;

    // Command Output
    // Fill in attest common fields
    FillInAttestInfo(in->signHandle, &in->inScheme, &in->qualifyingData, &timeInfo);

    // GetClock specific fields
    timeInfo.type                     = TPM_ST_ATTEST_TIME;
    timeInfo.attested.time.time.time = g_time;
    TimeFillInfo(&timeInfo.attested.time.time.clockInfo);

    // Firmware version in plain text
    timeInfo.attested.time.firmwareVersion =
        (((UINT64)gp.firmwareV1) << 32) + gp.firmwareV2;

    // Sign attestation structure.  A NULL signature will be returned if
    // signObject is NULL.
    return SignAttestInfo(signObject,
                          &in->inScheme,
                          &timeInfo,
                          &in->qualifyingData,
                          &out->timeInfo,
                          &out->signature);
}

#endif  // CC_GetTime
```

7.21 /tpm/src/command/Attestation/Quote.c

```
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "Quote_fp.h"

#if CC_Quote  // Conditional expansion of this file

/*(See part 3 specification)
// quote PCR values
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY              'signHandle' does not reference a signing key;
//      TPM_RC_SCHEME           the scheme is not compatible with sign key type,
//                              or input scheme is not compatible with default
//                              scheme, or the chosen scheme is not a valid
//                              sign scheme
TPM_RC
TPM2_Quote(Quote_In*  in,  // IN: input parameter list
           Quote_Out* out  // OUT: output parameter list
)
{
    TPMI_ALG_HASH hashAlg;
    TPMS_ATTEST   quoted;
    OBJECT*       signObject = HandleToObject(in->signHandle);
    // Input Validation
    if(!IsSigningObject(signObject))
```

```
        return TPM_RCS_KEY + RC_Quote_signHandle;
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_Quote_inScheme;

    // Command Output

    // Filling in attest information
    // Common fields
    // FillInAttestInfo may return TPM_RC_SCHEME or TPM_RC_KEY
    FillInAttestInfo(in->signHandle, &in->inScheme, &in->qualifyingData, &quoted);

    // Quote specific fields
    // Attestation type
    quoted.type = TPM_ST_ATTEST_QUOTE;

    // Get hash algorithm in sign scheme.  This hash algorithm is used to
    // compute PCR digest. If there is no algorithm, then the PCR cannot
    // be digested and this command returns TPM_RC_SCHEME
    hashAlg = in->inScheme.details.any.hashAlg;

    if(hashAlg == TPM_ALG_NULL)
        return TPM_RCS_SCHEME + RC_Quote_inScheme;

    // Compute PCR digest
    PCRComputeCurrentDigest(
        hashAlg, &in->PCRselect, &quoted.attested.quote.pcrDigest);

    // Copy PCR select.  "PCRselect" is modified in PCRComputeCurrentDigest
    // function
    quoted.attested.quote.pcrSelect = in->PCRselect;

    // Sign attestation structure.  A NULL signature will be returned if
    // signObject is NULL.
    return SignAttestInfo(signObject,
                          &in->inScheme,
                          &quoted,
                          &in->qualifyingData,
                          &out->quoted,
                          &out->signature);
}

#endif  // CC_Quote
```

7.22 /tpm/src/command/Capability/GetCapability.c

```
#include "Tpm.h"
#include "GetCapability_fp.h"

#if CC_GetCapability  // Conditional expansion of this file

/*(See part 3 specification)
// This command returns various information regarding the TPM and its current
// state
*/
//  Return Type: TPM_RC
//      TPM_RC_HANDLE       value of 'property' is in an unsupported handle range
//                          for the TPM_CAP_HANDLES 'capability' value
//      TPM_RC_VALUE        invalid 'capability'; or 'property' is not 0 for the
//                          TPM_CAP_PCRS 'capability' value
TPM_RC
TPM2_GetCapability(GetCapability_In*  in,  // IN: input parameter list
                   GetCapability_Out* out  // OUT: output parameter list
)
{
    TPMU_CAPABILITIES* data = &out->capabilityData.data;
```

```
    // Command Output

    // Set output capability type the same as input type
    out->capabilityData.capability = in->capability;

    switch(in->capability)
    {
        case TPM_CAP_ALGS:
            out->moreData = AlgorithmCapGetImplemented(
                (TPM_ALG_ID)in->property, in->propertyCount, &data->algorithms);
            break;
        case TPM_CAP_HANDLES:
            switch(HandleGetType((TPM_HANDLE)in->property))
            {
                case TPM_HT_TRANSIENT:
                    // Get list of handles of loaded transient objects
                    out->moreData = ObjectCapGetLoaded(
                        (TPM_HANDLE)in->property, in->propertyCount, &data->handles);
                    break;
                case TPM_HT_PERSISTENT:
                    // Get list of handles of persistent objects
                    out->moreData = NvCapGetPersistent(
                        (TPM_HANDLE)in->property, in->propertyCount, &data->handles);
                    break;
                case TPM_HT_NV_INDEX:
                    // Get list of defined NV index
                    out->moreData = NvCapGetIndex(
                        (TPM_HANDLE)in->property, in->propertyCount, &data->handles);
                    break;
                case TPM_HT_LOADED_SESSION:
                    // Get list of handles of loaded sessions
                    out->moreData = SessionCapGetLoaded(
                        (TPM_HANDLE)in->property, in->propertyCount, &data->handles);
                    break;
                case TPM_HT_SAVED_SESSION:
                    // Get list of handles of
                    out->moreData = SessionCapGetSaved(
                        (TPM_HANDLE)in->property, in->propertyCount, &data->handles);
                    break;
                case TPM_HT_PCR:
                    // Get list of handles of PCR
                    out->moreData = PCRCapGetHandles(
                        (TPM_HANDLE)in->property, in->propertyCount, &data->handles);
                    break;
                case TPM_HT_PERMANENT:
                    // Get list of permanent handles
                    out->moreData = PermanentCapGetHandles(
                        (TPM_HANDLE)in->property, in->propertyCount, &data->handles);
                    break;
                default:
                    // Unsupported input handle type
                    return TPM_RCS_HANDLE + RC_GetCapability_property;
                    break;
            }
            break;
        case TPM_CAP_COMMANDS:
            out->moreData = CommandCapGetCCList(
                (TPM_CC)in->property, in->propertyCount, &data->command);
            break;
        case TPM_CAP_PP_COMMANDS:
            out->moreData = PhysicalPresenceCapGetCCList(
                (TPM_CC)in->property, in->propertyCount, &data->ppCommands);
            break;
        case TPM_CAP_AUDIT_COMMANDS:
            out->moreData = CommandAuditCapGetCCList(
                (TPM_CC)in->property, in->propertyCount, &data->auditCommands);
```

```
            break;
        case TPM_CAP_PCRS:
            // Input property must be 0
            if(in->property != 0)
                return TPM_RCS_VALUE + RC_GetCapability_property;
            out->moreData =
                PCRCapGetAllocation(in->propertyCount, &data->assignedPCR);
            break;
        case TPM_CAP_PCR_PROPERTIES:
            out->moreData = PCRCapGetProperties(
                (TPM_PT_PCR)in->property, in->propertyCount, &data->pcrProperties);
            break;
        case TPM_CAP_TPM_PROPERTIES:
            out->moreData = TPMCapGetProperties(
                (TPM_PT)in->property, in->propertyCount, &data->tpmProperties);
            break;
#   if ALG_ECC
        case TPM_CAP_ECC_CURVES:
            out->moreData = CryptCapGetECCCurve(
                (TPM_ECC_CURVE)in->property, in->propertyCount, &data->eccCurves);
            break;
#   endif  // ALG_ECC
        case TPM_CAP_AUTH_POLICIES:
            if(HandleGetType((TPM_HANDLE)in->property) != TPM_HT_PERMANENT)
                return TPM_RCS_VALUE + RC_GetCapability_property;
            out->moreData = PermanentHandleGetPolicy(
                (TPM_HANDLE)in->property, in->propertyCount, &data->authPolicies);
            break;
        case TPM_CAP_ACT:
#   if ACT_SUPPORT
            if(((TPM_RH)in->property < TPM_RH_ACT_0)
               || ((TPM_RH)in->property > TPM_RH_ACT_F))
                return TPM_RCS_VALUE + RC_GetCapability_property;
            out->moreData = ActGetCapabilityData(
                (TPM_HANDLE)in->property, in->propertyCount, &data->actData);
            break;
#   else
            return TPM_RCS_VALUE + RC_GetCapability_property;
#   endif  // ACT_SUPPORT
        case TPM_CAP_VENDOR_PROPERTY:
            // vendor property is not implemented
        default:
            // Unsupported TPM_CAP value
            return TPM_RCS_VALUE + RC_GetCapability_capability;
            break;
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_GetCapability
```

7.23 /tpm/src/command/Capability/SetCapability.c

```
#include "Tpm.h"
#include "SetCapability_fp.h"

#if CC_SetCapability  // Conditional expansion of this file

/*(See part 3 specification)
// This command allows configuration of the TPM's capabilities.
*/
//  Return Type: TPM_RC
//      TPM_RC_HANDLE       value of 'property' is in an unsupported handle range
//                          for the TPM_CAP_HANDLES 'capability' value
```

```
//      TPM_RC_VALUE        invalid 'capability'
TPM_RC
TPM2_SetCapability(SetCapability_In* in  // IN: input parameter list
)
{
    // This reference implementation does not implement any settable capabilities.
    return TPM_RCS_VALUE + SetCapability_setCapabilityData;
}

#endif  // CC_SetCapability
```

7.24 /tpm/src/command/Capability/TestParms.c

```
#include "Tpm.h"
#include "TestParms_fp.h"

#if CC_TestParms  // Conditional expansion of this file

/*(See part 3 specification)
// TestParms
*/
TPM_RC
TPM2_TestParms(TestParms_In* in  // IN: input parameter list
)
{
    // Input parameter is not reference in command action
    NOT_REFERENCED(in);

    // The parameters are tested at unmarshal process.  We do nothing in command
    // action
    return TPM_RC_SUCCESS;
}

#endif  // CC_TestParms
```

7.25 /tpm/src/command/ClockTimer/ACT_SetTimeout.c

```
#include "Tpm.h"
#include "ACT_SetTimeout_fp.h"

#if CC_ACT_SetTimeout  // Conditional expansion of this file

/*(See part 3 specification)
// prove an object with a specific Name is loaded in the TPM
*/
//  Return Type: TPM_RC
//      TPM_RC_RETRY            returned when an update for the selected ACT is
//                              already pending
//      TPM_RC_VALUE            attempt to disable signaling from an ACT that has
//                              not expired
TPM_RC
TPM2_ACT_SetTimeout(ACT_SetTimeout_In* in  // IN: input parameter list
)
{
    // If 'startTimeout' is UINT32_MAX, then this is an attempt to disable the ACT
    // and turn off the signaling for the ACT. This is only valid if the ACT
    // is signaling.
#  if ACT_SUPPORT
    if((in->startTimeout == UINT32_MAX) && !ActGetSignaled(in->actHandle))
        return TPM_RC_VALUE + RC_ACT_SetTimeout_startTimeout;
    return ActCounterUpdate(in->actHandle, in->startTimeout);
#  else   // ACT_SUPPORT
    NOT_REFERENCED(in);
    return TPM_RC_VALUE + RC_ACT_SetTimeout_startTimeout;
```

```
#  endif  // ACT_SUPPORT
}

#endif  // CC_ACT_SetTimeout
```

7.26 /tpm/src/command/ClockTimer/ACT_spt.c

```
//** Introduction
// This code implements the ACT update code. It does not use a mutex. This code uses
// a platform service (_plat__ACT_UpdateCounter()) that returns 'false' if the update
// is not accepted. If this occurs, then TPM_RC_RETRY should be sent to the caller so
// that they can retry the operation later. The implementation of this is platform
// dependent but the reference uses a simple flag to indicate that an update is
// pending and the only process that can clear that flag is the process that does the
// actual update.

//** Includes
#include "Tpm.h"
#include "ACT_spt_fp.h"
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/tpm_to_platform_interface.h>

//** Functions

#if ACT_SUPPORT

//*** _ActResume()
// This function does the resume processing for an ACT. It updates the saved count
// and turns signaling back on if necessary.
static void _ActResume(UINT32     act,     //IN: the act number
                       ACT_STATE* actData  //IN: pointer to the saved ACT data
)
{
    // If the act was non-zero, then restore the counter value.
    if(actData->remaining > 0)
        _plat__ACT_UpdateCounter(act, actData->remaining);
    // if the counter was zero and the ACT signaling, enable the signaling.
    else if(go.signaledACT & (1 << act))
        _plat__ACT_SetSignaled(act, TRUE);
}

//*** ActStartup()
// This function is called by TPM2_Startup() to initialize the ACT counter values.
BOOL ActStartup(STARTUP_TYPE type)
{
    // Reset all the ACT hardware
    _plat__ACT_Initialize();

    // If this not a cold start, copy all the current 'signaled' settings to
    // 'preservedSignaled'.
    if(g_powerWasLost)
        go.preservedSignaled = 0;
    else
        go.preservedSignaled |= go.signaledACT;

    // For TPM_RESET or TPM_RESTART, the ACTs will all be disabled and the output
    // de-asserted.
    if(type != SU_RESUME)
    {
        go.signaledACT = 0;
#   define CLEAR_ACT_POLICY(N)                          \
      go.ACT_##N.hashAlg           = TPM_ALG_NULL; \
      go.ACT_##N.authPolicy.b.size = 0;
      FOR_EACH_ACT(CLEAR_ACT_POLICY)
```

```
    }
    else
    {
        // Resume each of the implemented ACT
#   define RESUME_ACT(N) _ActResume(0x##N, &go.ACT_##N);

        FOR_EACH_ACT(RESUME_ACT)
    }
    // set no ACT updated since last startup. This is to enable the halving of the
    // timeout value
    s_ActUpdated = 0;
    _plat__ACT_EnableTicks(TRUE);
    return TRUE;
}

//*** _ActSaveState()
// Get the counter state and the signaled state for an ACT. If the ACT has not been
// updated since the last time it was saved, then divide the count by 2.
static void _ActSaveState(UINT32 act, P_ACT_STATE actData)
{
    actData->remaining = _plat__ACT_GetRemaining(act);
    // If the ACT hasn't been updated since the last startup, then it should be
    // be halved.
    if((s_ActUpdated & (1 << act)) == 0)
    {
        // Don't halve if the count is set to max or if halving would make it zero
        if((actData->remaining != UINT32_MAX) && (actData->remaining > 1))
            actData->remaining /= 2;
    }
    if(_plat__ACT_GetSignaled(act))
        go.signaledACT |= (1 << act);
}

//*** ActGetSignaled()
// This function returns the state of the signaled flag associated with an ACT.
BOOL ActGetSignaled(TPM_RH actHandle)
{
    UINT32 act = actHandle - TPM_RH_ACT_0;
    //
    return _plat__ACT_GetSignaled(act);
}

//***ActShutdown()
// This function saves the current state of the counters
BOOL ActShutdown(TPM_SU state  //IN: the type of the shutdown.
)
{
    // if this is not shutdown state, then the only type of startup is TPM_RESTART
    // so the timer values will be cleared. If this is shutdown state, get the current
    // countdown and signaled values. Plus, if the counter has not been updated
    // since the last restart, divide the time by 2 so that there is no attack on the
    // countdown by saving the countdown state early and then not using the TPM.
    if(state == TPM_SU_STATE)
    {
        // This will be populated as each of the ACT is queried
        go.signaledACT = 0;
        // Get the current count and the signaled state
#   define SAVE_ACT_STATE(N) _ActSaveState(0x##N, &go.ACT_##N);

        FOR_EACH_ACT(SAVE_ACT_STATE);
    }
    return TRUE;
}

//*** ActIsImplemented()
// This function determines if an ACT is implemented in both the TPM and the platform
```

```
// code.
BOOL ActIsImplemented(UINT32 act)
{
    // This switch accounts for the TPM implemented values.
    switch(act)
    {
        FOR_EACH_ACT(CASE_ACT_NUMBER)
        // This ensures that the platform implements the values implemented by
        // the TPM
        return _plat__ACT_GetImplemented(act);
        default:
            break;
    }
    return FALSE;
}

//***ActCounterUpdate()
// This function updates the ACT counter. If the counter already has a pending update,
// it returns TPM_RC_RETRY so that the update can be tried again later.
TPM_RC
ActCounterUpdate(TPM_RH handle,   //IN: the handle of the act
                 UINT32 newValue  //IN: the value to set in the ACT
)
{
    UINT32 act;
    TPM_RC result;
    //
    act = handle - TPM_RH_ACT_0;
    // This should never fail, but...
    if(!_plat__ACT_GetImplemented(act))
        result = TPM_RC_VALUE;
    else
    {
        // Will need to clear orderly so fail if we are orderly and NV is
        // not available
        if(NV_IS_ORDERLY)
            RETURN_IF_NV_IS_NOT_AVAILABLE;
        // if the attempt to update the counter fails, it means that there is an
        // update pending so wait until it has occurred and then do an update.
        if(!_plat__ACT_UpdateCounter(act, newValue))
            result = TPM_RC_RETRY;
        else
        {
            // Indicate that the ACT has been updated since last TPM2_Startup().
            s_ActUpdated |= (UINT16)(1 << act);

            // Clear the preservedSignaled attribute.
            go.preservedSignaled &= ~((UINT16)(1 << act));

            // Need to clear the orderly flag
            g_clearOrderly = TRUE;

            result         = TPM_RC_SUCCESS;
        }
    }
    return result;
}

//*** ActGetCapabilityData()
// This function returns the list of ACT data
//  Return Type: TPMI_YES_NO
//      YES             if more ACT data is available
//      NO              if no more ACT data to
TPMI_YES_NO
ActGetCapabilityData(TPM_HANDLE     actHandle,  // IN: the handle for the starting ACT
                     UINT32         maxCount,   // IN: maximum allowed return values
```

```
                    TPML_ACT_DATA* actList     // OUT: ACT data list
)
{
    // Initialize output property list
    actList->count = 0;

    // Make sure that the starting handle value is in range (again)
    if((actHandle < TPM_RH_ACT_0) || (actHandle > TPM_RH_ACT_F))
        return FALSE;
    // The maximum count of curves we may return is MAX_ECC_CURVES
    if(maxCount > MAX_ACT_DATA)
        maxCount = MAX_ACT_DATA;
    // Scan the ACT data from the starting ACT
    for(; actHandle <= TPM_RH_ACT_F; actHandle++)
    {
        UINT32 act = actHandle - TPM_RH_ACT_0;
        if(actList->count < maxCount)
        {
            if(ActIsImplemented(act))
            {
                TPMS_ACT_DATA* actData = &actList->actData[actList->count];
                //
                memset(&actData->attributes, 0, sizeof(actData->attributes));
                actData->handle  = actHandle;
                actData->timeout = _plat__ACT_GetRemaining(act);
                if(_plat__ACT_GetSignaled(act))
                    SET_ATTRIBUTE(actData->attributes, TPMA_ACT, signaled);
                else
                    CLEAR_ATTRIBUTE(actData->attributes, TPMA_ACT, signaled);
                if(go.preservedSignaled & (1 << act))
                    SET_ATTRIBUTE(actData->attributes, TPMA_ACT, preserveSignaled);
                actList->count++;
            }
        }
        else
        {
            if(_plat__ACT_GetImplemented(act))
                return YES;
        }
    }
    // If we get here, either all of the ACT values were put in the list, or the list
    // was filled and there are no more ACT values to return
    return NO;
}

//*** ActGetOneCapability()
// This function returns an ACT's capability, if present.
BOOL ActGetOneCapability(TPM_HANDLE     actHandle,  // IN: the handle for the ACT
                         TPMS_ACT_DATA* actData     // OUT: ACT data
)
{
    UINT32 act = actHandle - TPM_RH_ACT_0;

    if(ActIsImplemented(actHandle - TPM_RH_ACT_0))
    {
        memset(&actData->attributes, 0, sizeof(actData->attributes));
        actData->handle  = actHandle;
        actData->timeout = _plat__ACT_GetRemaining(act);
        if(_plat__ACT_GetSignaled(act))
            SET_ATTRIBUTE(actData->attributes, TPMA_ACT, signaled);
        else
            CLEAR_ATTRIBUTE(actData->attributes, TPMA_ACT, signaled);
        if(go.preservedSignaled & (1 << act))
            SET_ATTRIBUTE(actData->attributes, TPMA_ACT, preserveSignaled);
        return TRUE;
    }
```

```
    return FALSE;
}

#endif  // ACT_SUPPORT
```

7.27 /tpm/src/command/ClockTimer/ClockRateAdjust.c

```
#include "Tpm.h"
#include "ClockRateAdjust_fp.h"

#if CC_ClockRateAdjust  // Conditional expansion of this file

/*(See part 3 specification)
// adjusts the rate of advance of Clock and Timer to provide a better
// approximation to real time.
*/
TPM_RC
TPM2_ClockRateAdjust(ClockRateAdjust_In* in  // IN: input parameter list
)
{
    // Internal Data Update
    TimeSetAdjustRate(in->rateAdjust);

    return TPM_RC_SUCCESS;
}

#endif  // CC_ClockRateAdjust
```

7.28 /tpm/src/command/ClockTimer/ClockSet.c

```
#include "Tpm.h"
#include "ClockSet_fp.h"

#if CC_ClockSet  // Conditional expansion of this file

// Read the current TPMS_TIMER_INFO structure settings
//  Return Type: TPM_RC
//      TPM_RC_NV_RATE              NV is unavailable because of rate limit
//      TPM_RC_NV_UNAVAILABLE       NV is inaccessible
//      TPM_RC_VALUE                invalid new clock

TPM_RC
TPM2_ClockSet(ClockSet_In* in  // IN: input parameter list
)
{
    // Input Validation
    // new time can not be bigger than 0xFFFF000000000000 or smaller than
    // current clock
    if(in->newTime > 0xFFFF000000000000ULL || in->newTime < go.clock)
        return TPM_RCS_VALUE + RC_ClockSet_newTime;

    // Internal Data Update
    // Can't modify the clock if NV is not available.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    TimeClockUpdate(in->newTime);
    return TPM_RC_SUCCESS;
}

#endif  // CC_ClockSet
```

7.29 /tpm/src/command/ClockTimer/ReadClock.c

```
#include "Tpm.h"
#include "ReadClock_fp.h"

#if CC_ReadClock  // Conditional expansion of this file

/*(See part 3 specification)
// read the current TPMS_TIMER_INFO structure settings
*/
TPM_RC
TPM2_ReadClock(ReadClock_Out* out  // OUT: output parameter list
)
{
    // Command Output

    out->currentTime.time = g_time;
    TimeFillInfo(&out->currentTime.clockInfo);

    return TPM_RC_SUCCESS;
}

#endif  // CC_ReadClock
```

7.30 /tpm/src/command/CommandAudit/SetCommandCodeAuditStatus.c

```
#include "Tpm.h"
#include "SetCommandCodeAuditStatus_fp.h"

#if CC_SetCommandCodeAuditStatus  // Conditional expansion of this file

/*(See part 3 specification)
// change the audit status of a command or to set the hash algorithm used for
// the audit digest.
*/
TPM_RC
TPM2_SetCommandCodeAuditStatus(
    SetCommandCodeAuditStatus_In* in  // IN: input parameter list
)
{

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Internal Data Update

    // Update hash algorithm
    if(in->auditAlg != TPM_ALG_NULL && in->auditAlg != gp.auditHashAlg)
    {
        // Can't change the algorithm and command list at the same time
        if(in->setList.count != 0 || in->clearList.count != 0)
            return TPM_RCS_VALUE + RC_SetCommandCodeAuditStatus_auditAlg;

        // Change the hash algorithm for audit
        gp.auditHashAlg = in->auditAlg;

        // Set the digest size to a unique value that indicates that the digest
        // algorithm has been changed. The size will be cleared to zero in the
        // command audit processing on exit.
        gr.commandAuditDigest.t.size = 1;

        // Save the change of command audit data (this sets g_updateNV so that NV
        // will be updated on exit.)
```

```
        NV_SYNC_PERSISTENT(auditHashAlg);
    }
    else
    {
        UINT32 i;
        BOOL   changed = FALSE;

        // Process set list
        for(i = 0; i < in->setList.count; i++)

            // If change is made in CommandAuditSet, set changed flag
            if(CommandAuditSet(in->setList.commandCodes[i]))
                changed = TRUE;

        // Process clear list
        for(i = 0; i < in->clearList.count; i++)
            // If change is made in CommandAuditClear, set changed flag
            if(CommandAuditClear(in->clearList.commandCodes[i]))
                changed = TRUE;

        // if change was made to command list, update NV
        if(changed)
            // this sets g_updateNV so that NV will be updated on exit.
            NV_SYNC_PERSISTENT(auditCommands);
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_SetCommandCodeAuditStatus
```

7.31 /tpm/src/command/Context/ContextLoad.c

```
#include "Tpm.h"

#if CC_ContextLoad  // Conditional expansion of this file

#  include "ContextLoad_fp.h"
#  include "Marshal.h"
#  include "Context_spt_fp.h"

/*(See part 3 specification)
// Load context
*/

//  Return Type: TPM_RC
//      TPM_RC_CONTEXT_GAP          there is only one available slot and this is not
//                                  the oldest saved session context
//      TPM_RC_HANDLE               'context.savedHandle' does not reference a saved
//                                  session
//      TPM_RC_HIERARCHY            'context.hierarchy' is disabled
//      TPM_RC_INTEGRITY            'context' integrity check fail
//      TPM_RC_OBJECT_MEMORY        no free slot for an object
//      TPM_RC_SESSION_MEMORY       no free session slots
//      TPM_RC_SIZE                 incorrect context blob size
TPM_RC
TPM2_ContextLoad(ContextLoad_In*  in,  // IN: input parameter list
                 ContextLoad_Out* out  // OUT: output parameter list
)
{
    TPM_RC        result;
    TPM2B_DIGEST  integrityToCompare;
    TPM2B_DIGEST  integrity;
    BYTE*         buffer;  // defined to save some typing
    INT32         size;    // defined to save some typing
```

```
    TPM_HT        handleType;
    TPM2B_SYM_KEY symKey;
    TPM2B_IV      iv;

    // Input Validation

    // See discussion about the context format in TPM2_ContextSave Detailed Actions

    // IF this is a session context, make sure that the sequence number is
    // consistent with the version in the slot

    // Check context blob size
    handleType = HandleGetType(in->context.savedHandle);

    // Get integrity from context blob
    buffer = in->context.contextBlob.t.buffer;
    size   = (INT32)in->context.contextBlob.t.size;
    result = TPM2B_DIGEST_Unmarshal(&integrity, &buffer, &size);
    if(result != TPM_RC_SUCCESS)
        return result;

    // the size of the integrity value has to match the size of digest produced
    // by the integrity hash
    if(integrity.t.size != CryptHashGetDigestSize(CONTEXT_INTEGRITY_HASH_ALG))
        return TPM_RCS_SIZE + RC_ContextLoad_context;

    // Make sure that the context blob has enough space for the fingerprint. This
    // is elastic pants to go with the belt and suspenders we already have to make
    // sure that the context is complete and untampered.
    if((unsigned)size < sizeof(in->context.sequence))
        return TPM_RCS_SIZE + RC_ContextLoad_context;

    // After unmarshaling the integrity value, 'buffer' is pointing at the first
    // byte of the integrity protected and encrypted buffer and 'size' is the number
    // of integrity protected and encrypted bytes.

    // Compute context integrity
    result = ComputeContextIntegrity(&in->context, &integrityToCompare);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Compare integrity
    if(!MemoryEqual2B(&integrity.b, &integrityToCompare.b))
        return TPM_RCS_INTEGRITY + RC_ContextLoad_context;
    // Compute context encryption key
    result = ComputeContextProtectionKey(&in->context, &symKey, &iv);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Decrypt context data in place
    CryptSymmetricDecrypt(buffer,
                          CONTEXT_ENCRYPT_ALG,
                          CONTEXT_ENCRYPT_KEY_BITS,
                          symKey.t.buffer,
                          &iv,
                          TPM_ALG_CFB,
                          size,
                          buffer);
    // See if the fingerprint value matches. If not, it is symptomatic of either
    // a broken TPM or that the TPM is under attack so go into failure mode.
    if(!MemoryEqual(buffer, &in->context.sequence, sizeof(in->context.sequence)))
        FAIL(FATAL_ERROR_INTERNAL);

    // step over fingerprint
    buffer += sizeof(in->context.sequence);
```

```
    // set the remaining size of the context
    size -= sizeof(in->context.sequence);

    // Perform object or session specific input check
    switch(handleType)
    {
        case TPM_HT_TRANSIENT:
        {
            OBJECT* outObject;

            if(size > (INT32)sizeof(OBJECT))
                FAIL(FATAL_ERROR_INTERNAL);

            // Discard any changes to the handle that the TRM might have made
            in->context.savedHandle = TRANSIENT_FIRST;

            // If hierarchy is disabled, no object context can be loaded in this
            // hierarchy
            if(!HierarchyIsEnabled(in->context.hierarchy))
                return TPM_RCS_HIERARCHY + RC_ContextLoad_context;

            // Restore object. If there is no empty space, indicate as much
            outObject =
                ObjectContextLoad((ANY_OBJECT_BUFFER*)buffer, &out->loadedHandle);
            if(outObject == NULL)
                return TPM_RC_OBJECT_MEMORY;

            break;
        }
        case TPM_HT_POLICY_SESSION:
        case TPM_HT_HMAC_SESSION:
        {
            if(size != sizeof(SESSION))
                FAIL(FATAL_ERROR_INTERNAL);

            // This command may cause the orderlyState to be cleared due to
            // the update of state reset data.  If this is the case, check if NV is
            // available first
            RETURN_IF_ORDERLY;

            // Check if input handle points to a valid saved session and that the
            // sequence number makes sense
            if(!SequenceNumberForSavedContextIsValid(&in->context))
                return TPM_RCS_HANDLE + RC_ContextLoad_context;

            // Restore session.  A TPM_RC_SESSION_MEMORY, TPM_RC_CONTEXT_GAP error
            // may be returned at this point
            result =
                SessionContextLoad((SESSION_BUF*)buffer, &in->context.savedHandle);
            if(result != TPM_RC_SUCCESS)
                return result;

            out->loadedHandle = in->context.savedHandle;

            // orderly state should be cleared because of the update of state
            // reset and state clear data
            g_clearOrderly = TRUE;

            break;
        }
        default:
            // Context blob may only have an object handle or a session handle.
            // All the other handle type should be filtered out at unmarshal
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
```

```
    return TPM_RC_SUCCESS;
}

#endif  // CC_ContextLoad
```

7.32 /tpm/src/command/Context/ContextSave.c

```
#include "Tpm.h"

#if CC_ContextSave  // Conditional expansion of this file

#  include "ContextSave_fp.h"
#  include "Marshal.h"
#  include "Context_spt_fp.h"

/*(See part 3 specification)
 Save context
*/
//  Return Type: TPM_RC
//      TPM_RC_CONTEXT_GAP          a contextID could not be assigned for a session
//                                  context save
//      TPM_RC_TOO_MANY_CONTEXTS    no more contexts can be saved as the counter has
//                                  maxed out
TPM_RC
TPM2_ContextSave(ContextSave_In*  in,  // IN: input parameter list
                 ContextSave_Out* out  // OUT: output parameter list
)
{
    TPM_RC result = TPM_RC_SUCCESS;
    UINT16 fingerprintSize;  // The size of fingerprint in context
    // blob.
    UINT64        contextID = 0;  // session context ID
    TPM2B_SYM_KEY symKey;
    TPM2B_IV      iv;

    TPM2B_DIGEST  integrity;
    UINT16        integritySize;
    BYTE*         buffer;

    // This command may cause the orderlyState to be cleared due to
    // the update of state reset data. If the state is orderly and
    // cannot be changed, exit early.
    RETURN_IF_ORDERLY;

    // Internal Data Update

    // This implementation does not do things in quite the same way as described in
    // Part 2 of the specification. In Part 2, it indicates that the
    // TPMS_CONTEXT_DATA contains two TPM2B values. That is not how this is
    // implemented. Rather, the size field of the TPM2B_CONTEXT_DATA is used to
    // determine the amount of data in the encrypted data. That part is not
    // independently sized. This makes the actual size 2 bytes smaller than
    // calculated using Part 2. Since this is opaque to the caller, it is not
    // necessary to fix. The actual size is returned by TPM2_GetCapabilties().

    // Initialize output handle.  At the end of command action, the output
    // handle of an object will be replaced, while the output handle
    // for a session will be the same as input
    out->context.savedHandle = in->saveHandle;

    // Get the size of fingerprint in context blob.  The sequence value in
    // TPMS_CONTEXT structure is used as the fingerprint
    fingerprintSize = sizeof(out->context.sequence);
```

```
    // Compute the integrity size at the beginning of context blob
    integritySize =
        sizeof(integrity.t.size) + CryptHashGetDigestSize(CONTEXT_INTEGRITY_HASH_ALG);

    // Perform object or session specific context save
    switch(HandleGetType(in->saveHandle))
    {
        case TPM_HT_TRANSIENT:
        {
            OBJECT*             object = HandleToObject(in->saveHandle);
            ANY_OBJECT_BUFFER* outObject;
            UINT16 objectSize = ObjectIsSequence(object) ? sizeof(HASH_OBJECT)
                                                         : sizeof(OBJECT);

            outObject         = (ANY_OBJECT_BUFFER*)(out->context.contextBlob.t.buffer
                                        + integritySize + fingerprintSize);

            // Set size of the context data.  The contents of context blob is vendor
            // defined.  In this implementation, the size is size of integrity
            // plus fingerprint plus the whole internal OBJECT structure
            out->context.contextBlob.t.size =
                integritySize + fingerprintSize + objectSize;
#  if ALG_RSA
            // For an RSA key, make sure that the key has had the private exponent
            // computed before saving.
            if(object->publicArea.type == TPM_ALG_RSA
               && !(object->attributes.publicOnly))
                CryptRsaLoadPrivateExponent(&object->publicArea, &object->sensitive);
#  endif
            // Make sure things fit
            pAssert(out->context.contextBlob.t.size
                    <= sizeof(out->context.contextBlob.t.buffer));
            // Copy the whole internal OBJECT structure to context blob
            MemoryCopy(outObject, object, objectSize);

            // Increment object context ID
            gr.objectContextID++;
            // If object context ID overflows, TPM should be put in failure mode
            if(gr.objectContextID == 0)
                FAIL(FATAL_ERROR_INTERNAL);

            // Fill in other return values for an object.
            out->context.sequence = gr.objectContextID;
            // For regular object, savedHandle is 0x80000000.  For sequence object,
            // savedHandle is 0x80000001.  For object with stClear, savedHandle
            // is 0x80000002
            if(ObjectIsSequence(object))
            {
                out->context.savedHandle = 0x80000001;
                SequenceDataExport((HASH_OBJECT*)object,
                                   (HASH_OBJECT_BUFFER*)outObject);
            }
            else
                out->context.savedHandle =
                    (object->attributes.stClear == SET) ? 0x80000002 : 0x80000000;
            // Get object hierarchy
            out->context.hierarchy = object->hierarchy;

            break;
        }
        case TPM_HT_HMAC_SESSION:
        case TPM_HT_POLICY_SESSION:
        {
            SESSION* session = SessionGet(in->saveHandle);

            // Set size of the context data.  The contents of context blob is vendor
```

```
            // defined.  In this implementation, the size of context blob is the
            // size of a internal session structure plus the size of
            // fingerprint plus the size of integrity
            out->context.contextBlob.t.size =
                integritySize + fingerprintSize + sizeof(*session);

            // Make sure things fit
            pAssert(out->context.contextBlob.t.size
                    < sizeof(out->context.contextBlob.t.buffer));

            // Copy the whole internal SESSION structure to context blob.
            // Save space for fingerprint at the beginning of the buffer
            // This is done before anything else so that the actual context
            // can be reclaimed after this call
            pAssert(sizeof(*session) <= sizeof(out->context.contextBlob.t.buffer)
                                            - integritySize - fingerprintSize);
            MemoryCopy(
                out->context.contextBlob.t.buffer + integritySize + fingerprintSize,
                session,
                sizeof(*session));
            // Fill in the other return parameters for a session
            // Get a context ID and set the session tracking values appropriately
            // TPM_RC_CONTEXT_GAP is a possible error.
            // SessionContextSave() will flush the in-memory context
            // so no additional errors may occur after this call.
            result = SessionContextSave(out->context.savedHandle, &contextID);
            if(result != TPM_RC_SUCCESS)
                return result;
            // sequence number is the current session contextID
            out->context.sequence = contextID;

            // use TPM_RH_NULL as hierarchy for session context
            out->context.hierarchy = TPM_RH_NULL;

            break;
        }
        default:
            // SaveContext may only take an object handle or a session handle.
            // All the other handle type should be filtered out at unmarshal
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }

    // Save fingerprint at the beginning of encrypted area of context blob.
    // Reserve the integrity space
    pAssert(sizeof(out->context.sequence)
            <= sizeof(out->context.contextBlob.t.buffer) - integritySize);
    MemoryCopy(out->context.contextBlob.t.buffer + integritySize,
               &out->context.sequence,
               sizeof(out->context.sequence));

    // Compute context encryption key
    result = ComputeContextProtectionKey(&out->context, &symKey, &iv);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Encrypt context blob
    CryptSymmetricEncrypt(out->context.contextBlob.t.buffer + integritySize,
                          CONTEXT_ENCRYPT_ALG,
                          CONTEXT_ENCRYPT_KEY_BITS,
                          symKey.t.buffer,
                          &iv,
                          TPM_ALG_CFB,
                          out->context.contextBlob.t.size - integritySize,
                          out->context.contextBlob.t.buffer + integritySize);
```

```
    // Compute integrity hash for the object
    // In this implementation, the same routine is used for both sessions
    // and objects.
    result = ComputeContextIntegrity(&out->context, &integrity);
    if(result != TPM_RC_SUCCESS)
        return result;

    // add integrity at the beginning of context blob
    buffer = out->context.contextBlob.t.buffer;
    TPM2B_DIGEST_Marshal(&integrity, &buffer, NULL);

    // orderly state should be cleared because of the update of state reset and
    // state clear data
    g_clearOrderly = TRUE;

    return result;
}

#endif  // CC_ContextSave
```

7.33 /tpm/src/command/Context/Context_spt.c

```
//** Includes

#include "Tpm.h"
#include "Context_spt_fp.h"

//** Functions

//*** ComputeContextProtectionKey()
// This function retrieves the symmetric protection key for context encryption
// It is used by TPM2_ConextSave and TPM2_ContextLoad to create the symmetric
// encryption key and iv
/*(See part 1 specification)
    KDFa is used to generate the symmetric encryption key and IV. The parameters
    of the call are:
        Symkey = KDFa(hashAlg, hProof, vendorString, sequence, handle, bits)
    where
    hashAlg         a vendor-defined hash algorithm
    hProof          the hierarchy proof as selected by the hierarchy parameter
                    of the TPMS_CONTEXT
    vendorString    a value used to differentiate the uses of the KDF
    sequence        the sequence parameter of the TPMS_CONTEXT
    handle          the handle parameter of the TPMS_CONTEXT
    bits            the number of bits needed for a symmetric key and IV for
                    the context encryption
*/
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM
//                              failed to derive the Firmware SVN Secret for the
//                              requested SVN.
TPM_RC ComputeContextProtectionKey(TPMS_CONTEXT*  contextBlob,  // IN: context blob
                                   TPM2B_SYM_KEY* symKey,  // OUT: the symmetric key
                                   TPM2B_IV*      iv       // OUT: the IV.
)
{
    TPM_RC result = TPM_RC_SUCCESS;
    UINT16 symKeyBits;  // number of bits in the parent's
                        //   symmetric key
    TPM2B_PROOF proof;  // the proof value to use
```

```
    BYTE        kdfResult[sizeof(TPMU_HA) * 2];  // Value produced by the KDF

    TPM2B_DATA  sequence2B, handle2B;

    // Get sequence value in 2B format
    sequence2B.t.size = sizeof(contextBlob->sequence);
    MUST_BE(sizeof(contextBlob->sequence) <= sizeof(sequence2B.t.buffer));
    MemoryCopy(sequence2B.t.buffer, &contextBlob->sequence, sequence2B.t.size);

    // Get handle value in 2B format
    handle2B.t.size = sizeof(contextBlob->savedHandle);
    MUST_BE(sizeof(contextBlob->savedHandle) <= sizeof(handle2B.t.buffer));
    MemoryCopy(handle2B.t.buffer, &contextBlob->savedHandle, handle2B.t.size);

    // Get the symmetric encryption key size
    symKey->t.size = CONTEXT_ENCRYPT_KEY_BYTES;
    symKeyBits     = CONTEXT_ENCRYPT_KEY_BITS;
    // Get the size of the IV for the algorithm
    iv->t.size = CryptGetSymmetricBlockSize(CONTEXT_ENCRYPT_ALG, symKeyBits);

    // Get proof value
    result = HierarchyGetProof(contextBlob->hierarchy, &proof);
    if(result != TPM_RC_SUCCESS)
        return result;

    // KDFa to generate symmetric key and IV value
    CryptKDFa(CONTEXT_INTEGRITY_HASH_ALG,
              &proof.b,
              CONTEXT_KEY,
              &sequence2B.b,
              &handle2B.b,
              (symKey->t.size + iv->t.size) * 8,
              kdfResult,
              NULL,
              FALSE);

    MemorySet(proof.b.buffer, 0, proof.b.size);

    // Copy part of the returned value as the key
    pAssert(symKey->t.size <= sizeof(symKey->t.buffer));
    MemoryCopy(symKey->t.buffer, kdfResult, symKey->t.size);

    // Copy the rest as the IV
    pAssert(iv->t.size <= sizeof(iv->t.buffer));
    MemoryCopy(iv->t.buffer, &kdfResult[symKey->t.size], iv->t.size);

    return TPM_RC_SUCCESS;
}

//*** ComputeContextIntegrity()
// Generate the integrity hash for a context
//       It is used by TPM2_ContextSave to create an integrity hash
//       and by TPM2_ContextLoad to compare an integrity hash
/*(See part 1 specification)
    The HMAC integrity computation for a saved context is:
    HMACvendorAlg(hProof, resetValue {|| clearCount} || sequence || handle ||
                encContext)
    where
    HMACvendorAlg       HMAC using a vendor-defined hash algorithm
    hProof              the hierarchy proof as selected by the hierarchy
                        parameter of the TPMS_CONTEXT
    resetValue          either a counter value that increments on each TPM Reset
                        and is not reset over the lifetime of the TPM or a random
                        value that changes on each TPM Reset and has the size of
                        the digest produced by vendorAlg
    clearCount          a counter value that is incremented on each TPM Reset
```

```
                        or TPM Restart. This value is only included if the handle
                        value is 0x80000002.
    sequence            the sequence parameter of the TPMS_CONTEXT
    handle              the handle parameter of the TPMS_CONTEXT
    encContext          the encrypted context blob
*/
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM
//                              failed to derive the Firmware SVN Secret for the
//                              requested SVN.
TPM_RC ComputeContextIntegrity(TPMS_CONTEXT* contextBlob,  // IN: context blob
                               TPM2B_DIGEST* integrity     // OUT: integrity
)
{
    TPM_RC      result = TPM_RC_SUCCESS;
    HMAC_STATE  hmacState;
    TPM2B_PROOF proof;
    UINT16      integritySize;

    // Get proof value
    result = HierarchyGetProof(contextBlob->hierarchy, &proof);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Start HMAC
    integrity->t.size =
        CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG, &proof.b);

    MemorySet(proof.b.buffer, 0, proof.b.size);

    // Compute integrity size at the beginning of context blob
    integritySize = sizeof(integrity->t.size) + integrity->t.size;

    // Adding total reset counter so that the context cannot be
    // used after a TPM Reset
    CryptDigestUpdateInt(
        &hmacState.hashState, sizeof(gp.totalResetCount), gp.totalResetCount);

    // If this is a ST_CLEAR object, add the clear count
    // so that this contest cannot be loaded after a TPM Restart
    if(contextBlob->savedHandle == 0x80000002)
        CryptDigestUpdateInt(
            &hmacState.hashState, sizeof(gr.clearCount), gr.clearCount);

    // Adding sequence number to the HMAC to make sure that it doesn't
    // get changed
    CryptDigestUpdateInt(
        &hmacState.hashState, sizeof(contextBlob->sequence), contextBlob->sequence);

    // Protect the handle
    CryptDigestUpdateInt(&hmacState.hashState,
                         sizeof(contextBlob->savedHandle),
                         contextBlob->savedHandle);

    // Adding sensitive contextData, skip the leading integrity area
    CryptDigestUpdate(&hmacState.hashState,
                      contextBlob->contextBlob.t.size - integritySize,
                      contextBlob->contextBlob.t.buffer + integritySize);

    // Complete HMAC
    CryptHmacEnd2B(&hmacState, &integrity->b);
```

```
    return TPM_RC_SUCCESS;
}

//*** SequenceDataExport();
// This function is used scan through the sequence object and
// either modify the hash state data for export (contextSave) or to
// import it into the internal format (contextLoad).
// This function should only be called after the sequence object has been copied
// to the context buffer (contextSave) or from the context buffer into the sequence
// object. The presumption is that the context buffer version of the data is the
// same size as the internal representation so nothing outsize of the hash context
// area gets modified.
void SequenceDataExport(
    HASH_OBJECT*        object,       // IN: an internal hash object
    HASH_OBJECT_BUFFER* exportObject  // OUT: a sequence context in a buffer
)
{
    // If the hash object is not an event, then only one hash context is needed
    int count = (object->attributes.eventSeq) ? HASH_COUNT : 1;

    for(count--; count >= 0; count--)
    {
        HASH_STATE* hash       = &object->state.hashState[count];
        size_t      offset     = (BYTE*)hash - (BYTE*)object;
        BYTE*       exportHash = &((BYTE*)exportObject)[offset];

        CryptHashExportState(hash, (EXPORT_HASH_STATE*)exportHash);
    }
}

//*** SequenceDataImport();
// This function is used scan through the sequence object and
// either modify the hash state data for export (contextSave) or to
// import it into the internal format (contextLoad).
// This function should only be called after the sequence object has been copied
// to the context buffer (contextSave) or from the context buffer into the sequence
// object. The presumption is that the context buffer version of the data is the
// same size as the internal representation so nothing outsize of the hash context
// area gets modified.
void SequenceDataImport(
    HASH_OBJECT*        object,       // IN/OUT: an internal hash object
    HASH_OBJECT_BUFFER* exportObject  // IN/OUT: a sequence context in a buffer
)
{
    // If the hash object is not an event, then only one hash context is needed
    int count = (object->attributes.eventSeq) ? HASH_COUNT : 1;

    for(count--; count >= 0; count--)
    {
        HASH_STATE* hash       = &object->state.hashState[count];
        size_t      offset     = (BYTE*)hash - (BYTE*)object;
        BYTE*       importHash = &((BYTE*)exportObject)[offset];
        //
        CryptHashImportState(hash, (EXPORT_HASH_STATE*)importHash);
    }
}
```

7.34 /tpm/src/command/Context/EvictControl.c

```
#include "Tpm.h"
#include "EvictControl_fp.h"

#if CC_EvictControl  // Conditional expansion of this file

/*(See part 3 specification)
```

```
// Make a transient object persistent or evict a persistent object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   an object with 'temporary', 'stClear' or 'publicOnly'
//                          attribute SET cannot be made persistent
//      TPM_RC_HIERARCHY    'auth' cannot authorize the operation in the hierarchy
//                          of 'evictObject';
//                          an object in a firmware-bound or SVN-bound hierarchy
//                          cannot be made persistent.
//      TPM_RC_HANDLE       'evictHandle' of the persistent object to be evicted is
//                          not the same as the 'persistentHandle' argument
//      TPM_RC_NV_HANDLE    'persistentHandle' is unavailable
//      TPM_RC_NV_SPACE     no space in NV to make 'evictHandle' persistent
//      TPM_RC_RANGE        'persistentHandle' is not in the range corresponding to
//                          the hierarchy of 'evictObject'
TPM_RC
TPM2_EvictControl(EvictControl_In* in  // IN: input parameter list
)
{
    TPM_RC  result;
    OBJECT* evictObject;

    // Input Validation

    // Get internal object pointer
    evictObject = HandleToObject(in->objectHandle);

    // Objects in a firmware-limited or SVN-limited hierarchy cannot be made
    // persistent.
    if(HierarchyIsFirmwareLimited(evictObject->hierarchy)
       || HierarchyIsSvnLimited(evictObject->hierarchy))
        return TPM_RCS_HIERARCHY + RC_EvictControl_objectHandle;

    // Temporary, stClear or public only objects can not be made persistent
    if(evictObject->attributes.temporary == SET
       || evictObject->attributes.stClear == SET
       || evictObject->attributes.publicOnly == SET)
        return TPM_RCS_ATTRIBUTES + RC_EvictControl_objectHandle;

    // If objectHandle refers to a persistent object, it should be the same as
    // input persistentHandle
    if(evictObject->attributes.evict == SET
       && evictObject->evictHandle != in->persistentHandle)
        return TPM_RCS_HANDLE + RC_EvictControl_objectHandle;

    // Additional authorization validation
    if(in->auth == TPM_RH_PLATFORM)
    {
        // To make persistent
        if(evictObject->attributes.evict == CLEAR)
        {
            // PlatformAuth can not set evict object in storage or endorsement
            // hierarchy
            if(evictObject->attributes.ppsHierarchy == CLEAR)
                return TPM_RCS_HIERARCHY + RC_EvictControl_objectHandle;
            // Platform cannot use a handle outside of platform persistent range.
            if(!NvIsPlatformPersistentHandle(in->persistentHandle))
                return TPM_RCS_RANGE + RC_EvictControl_persistentHandle;
        }
        // PlatformAuth can delete any persistent object
    }
    else if(in->auth == TPM_RH_OWNER)
    {
        // OwnerAuth can not set or clear evict object in platform hierarchy
        if(evictObject->attributes.ppsHierarchy == SET)
            return TPM_RCS_HIERARCHY + RC_EvictControl_objectHandle;
```

```
        // Owner cannot use a handle outside of owner persistent range.
        if(evictObject->attributes.evict == CLEAR
           && !NvIsOwnerPersistentHandle(in->persistentHandle))
            return TPM_RCS_RANGE + RC_EvictControl_persistentHandle;
    }
    else
    {
        // Other authorization is not allowed in this command and should have been
        // filtered out in unmarshal process
        FAIL(FATAL_ERROR_INTERNAL);
    }
    // Internal Data Update
    // Change evict state
    if(evictObject->attributes.evict == CLEAR)
    {
        // Make object persistent
        if(NvFindHandle(in->persistentHandle) != 0)
            return TPM_RC_NV_DEFINED;
        // A TPM_RC_NV_HANDLE or TPM_RC_NV_SPACE error may be returned at this
        // point
        result = NvAddEvictObject(in->persistentHandle, evictObject);
    }
    else
    {
        // Delete the persistent object in NV
        result = NvDeleteEvict(evictObject->evictHandle);
    }
    return result;
}

#endif  // CC_EvictControl
```

7.35 /tpm/src/command/Context/FlushContext.c

```
#include "Tpm.h"
#include "FlushContext_fp.h"

#if CC_FlushContext  // Conditional expansion of this file

/*(See part 3 specification)
// Flush a specific object or session
*/
//  Return Type: TPM_RC
//      TPM_RC_HANDLE       'flushHandle' does not reference a loaded object or session
TPM_RC
TPM2_FlushContext(FlushContext_In* in  // IN: input parameter list
)
{
    // Internal Data Update

    // Call object or session specific routine to flush
    switch(HandleGetType(in->flushHandle))
    {
        case TPM_HT_TRANSIENT:
            if(!IsObjectPresent(in->flushHandle))
                return TPM_RCS_HANDLE + RC_FlushContext_flushHandle;
            // Flush object
            FlushObject(in->flushHandle);
            break;
        case TPM_HT_HMAC_SESSION:
        case TPM_HT_POLICY_SESSION:
            if(!SessionIsLoaded(in->flushHandle) && !SessionIsSaved(in->flushHandle))
                return TPM_RCS_HANDLE + RC_FlushContext_flushHandle;
```

```
            // If the session to be flushed is the exclusive audit session, then
            // indicate that there is no exclusive audit session any longer.
            if(in->flushHandle == g_exclusiveAuditSession)
                g_exclusiveAuditSession = TPM_RH_UNASSIGNED;

            // Flush session
            SessionFlush(in->flushHandle);
            break;
        default:
            // This command only takes object or session handle.  Other handles
            // should be filtered out at handle unmarshal
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_FlushContext
```

7.36 /tpm/src/command/DA/DictionaryAttackLockReset.c

```
#include "Tpm.h"
#include "DictionaryAttackLockReset_fp.h"

#if CC_DictionaryAttackLockReset  // Conditional expansion of this file

/*(See part 3 specification)
// This command cancels the effect of a TPM lockout due to a number of
// successive authorization failures.  If this command is properly authorized,
// the lockout counter is set to 0.
*/
TPM_RC
TPM2_DictionaryAttackLockReset(
    DictionaryAttackLockReset_In* in  // IN: input parameter list
)
{
    // Input parameter is not reference in command action
    NOT_REFERENCED(in);

    // The command needs NV update.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Internal Data Update

    // Set failed tries to 0
    gp.failedTries = 0;

    // Record the changes to NV
    NV_SYNC_PERSISTENT(failedTries);

    return TPM_RC_SUCCESS;
}

#endif  // CC_DictionaryAttackLockReset
```

7.37 /tpm/src/command/DA/DictionaryAttackParameters.c

```
#include "Tpm.h"
#include "DictionaryAttackParameters_fp.h"

#if CC_DictionaryAttackParameters  // Conditional expansion of this file

/*(See part 3 specification)
```

```
// change the lockout parameters
*/
TPM_RC
TPM2_DictionaryAttackParameters(
    DictionaryAttackParameters_In* in  // IN: input parameter list
)
{
    // The command needs NV update.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Internal Data Update

    // Set dictionary attack parameters
    gp.maxTries        = in->newMaxTries;
    gp.recoveryTime    = in->newRecoveryTime;
    gp.lockoutRecovery = in->lockoutRecovery;

#  if 0
    // Errata eliminates this code
    // This functionality has been disabled. The preferred implementation is now
    // to leave failedTries unchanged when the parameters are changed. This could
    // have the effect of putting the TPM into DA lockout if in->newMaxTries is
    // not greater than the current value of gp.failedTries.
    // Set failed tries to 0
    gp.failedTries = 0;
#  endif

    // Record the changes to NV
    NV_SYNC_PERSISTENT(failedTries);
    NV_SYNC_PERSISTENT(maxTries);
    NV_SYNC_PERSISTENT(recoveryTime);
    NV_SYNC_PERSISTENT(lockoutRecovery);

    return TPM_RC_SUCCESS;
}

#endif  // CC_DictionaryAttackParameters
```

7.38 /tpm/src/command/Duplication/Duplicate.c

```
#include "Tpm.h"
#include "Duplicate_fp.h"

#if CC_Duplicate  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// Duplicate a loaded object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   key to duplicate has 'fixedParent' SET
//      TPM_RC_HASH         for an RSA key, the nameAlg digest size for the
//                          newParent is not compatible with the key size
//      TPM_RC_HIERARCHY    'encryptedDuplication' is SET and 'newParentHandle'
//                          specifies Null Hierarchy
//      TPM_RC_KEY          'newParentHandle' references invalid ECC key (public
//                          point not on the curve)
//      TPM_RC_SIZE         input encryption key size does not match the
//                          size specified in symmetric algorithm
//      TPM_RC_SYMMETRIC    'encryptedDuplication' is SET but no symmetric
//                          algorithm is provided
//      TPM_RC_TYPE         'newParentHandle' is neither a storage key nor
//                          TPM_RH_NULL; or the object has a NULL nameAlg
//      TPM_RC_VALUE        for an RSA newParent, the sizes of the digest and
```

```
//                          the encryption key are too large to be OAEP encoded
TPM_RC
TPM2_Duplicate(Duplicate_In*  in,  // IN: input parameter list
               Duplicate_Out* out  // OUT: output parameter list
)
{
    TPM_RC         result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE sensitive;

    UINT16         innerKeySize = 0;  // encrypt key size for inner wrap

    OBJECT*        object;
    OBJECT*        newParent;
    TPM2B_DATA     data;

    // Input Validation

    // Get duplicate object pointer
    object = HandleToObject(in->objectHandle);
    // Get new parent
    newParent = HandleToObject(in->newParentHandle);

    // duplicate key must have fixParent bit CLEAR.
    if(IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, fixedParent))
        return TPM_RCS_ATTRIBUTES + RC_Duplicate_objectHandle;

    // Do not duplicate object with NULL nameAlg
    if(object->publicArea.nameAlg == TPM_ALG_NULL)
        return TPM_RCS_TYPE + RC_Duplicate_objectHandle;

    // new parent key must be a storage object or TPM_RH_NULL
    if(in->newParentHandle != TPM_RH_NULL && !ObjectIsStorage(in->newParentHandle))
        return TPM_RCS_TYPE + RC_Duplicate_newParentHandle;

    // If the duplicated object has encryptedDuplication SET, then there must be
    // an inner wrapper and the new parent may not be TPM_RH_NULL
    if(IS_ATTRIBUTE(
           object->publicArea.objectAttributes, TPMA_OBJECT, encryptedDuplication))
    {
        if(in->symmetricAlg.algorithm == TPM_ALG_NULL)
            return TPM_RCS_SYMMETRIC + RC_Duplicate_symmetricAlg;
        if(in->newParentHandle == TPM_RH_NULL)
            return TPM_RCS_HIERARCHY + RC_Duplicate_newParentHandle;
    }

    if(in->symmetricAlg.algorithm == TPM_ALG_NULL)
    {
        // if algorithm is TPM_ALG_NULL, input key size must be 0
        if(in->encryptionKeyIn.t.size != 0)
            return TPM_RCS_SIZE + RC_Duplicate_encryptionKeyIn;
    }
    else
    {
        // Get inner wrap key size
        innerKeySize = in->symmetricAlg.keyBits.sym;

        // If provided the input symmetric key must match the size of the algorithm
        if(in->encryptionKeyIn.t.size != 0
           && in->encryptionKeyIn.t.size != (innerKeySize + 7) / 8)
            return TPM_RCS_SIZE + RC_Duplicate_encryptionKeyIn;
    }

    // Command Output

    if(in->newParentHandle != TPM_RH_NULL)
    {
```

```
        // Make encrypt key and its associated secret structure.  A TPM_RC_KEY
        // error may be returned at this point
        out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
        result =
            CryptSecretEncrypt(newParent, DUPLICATE_STRING, &data, &out->outSymSeed);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    else
    {
        // Do not apply outer wrapper
        data.t.size            = 0;
        out->outSymSeed.t.size = 0;
    }

    // Copy sensitive area
    sensitive = object->sensitive;

    // Prepare output private data from sensitive.
    // Note: If there is no encryption key, one will be provided by
    // SensitiveToDuplicate(). This is why the assignment of encryptionKeyIn to
    // encryptionKeyOut will work properly and is not conditional.
    SensitiveToDuplicate(&sensitive,
                         &object->name.b,
                         newParent,
                         object->publicArea.nameAlg,
                         &data.b,
                         &in->symmetricAlg,
                         &in->encryptionKeyIn,
                         &out->duplicate);

    out->encryptionKeyOut = in->encryptionKeyIn;

    return TPM_RC_SUCCESS;
}

#endif  // CC_Duplicate
```

7.39 /tpm/src/command/Duplication/Import.c

```
#include "Tpm.h"
#include "Import_fp.h"

#if CC_Import  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// This command allows an asymmetrically encrypted blob, containing a duplicated
// object to be re-encrypted using the group symmetric key associated with the
// parent.
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'FixedTPM' and 'fixedParent' of 'objectPublic' are not
//                              both CLEAR; or 'inSymSeed' is nonempty and
//                              'parentHandle' does not reference a decryption key; or
//                              'objectPublic' and 'parentHandle' have incompatible
//                              or inconsistent attributes; or
//                              encrytpedDuplication is SET in 'objectPublic' but the
//                              inner or outer wrapper is missing.
//                              Note that if the TPM provides parameter values, the
//                              parameter number will indicate 'symmetricKey' (missing
//                              inner wrapper) or 'inSymSeed' (missing outer wrapper)
//      TPM_RC_BINDING          'duplicate' and 'objectPublic' are not
//                              cryptographically bound
```

```
//      TPM_RC_ECC_POINT        'inSymSeed' is nonempty and ECC point in 'inSymSeed'
//                              is not on the curve
//      TPM_RC_HASH             'objectPublic' does not have a valid nameAlg
//      TPM_RC_INSUFFICIENT     'inSymSeed' is nonempty and failed to retrieve ECC
//                              point from the secret; or unmarshaling sensitive value
//                              from 'duplicate' failed the result of 'inSymSeed'
//                              decryption
//      TPM_RC_INTEGRITY        'duplicate' integrity is broken
//      TPM_RC_KDF              'objectPublic' representing decrypting keyed hash
//                              object specifies invalid KDF
//      TPM_RC_KEY              inconsistent parameters of 'objectPublic'; or
//                              'inSymSeed' is nonempty and 'parentHandle' does not
//                              reference a key of supported type; or
//                              invalid key size in 'objectPublic' representing an
//                              asymmetric key
//      TPM_RC_NO_RESULT        'inSymSeed' is nonempty and multiplication resulted in
//                              ECC point at infinity
//      TPM_RC_OBJECT_MEMORY    no available object slot
//      TPM_RC_SCHEME           inconsistent attributes 'decrypt', 'sign',
//                              'restricted' and key's scheme ID in 'objectPublic';
//                              or hash algorithm is inconsistent with the scheme ID
//                              for keyed hash object
//      TPM_RC_SIZE             'authPolicy' size does not match digest size of the
//                              name algorithm in 'objectPublic'; or
//                              'symmetricAlg' and 'encryptionKey' have different
//                              sizes; or
//                              'inSymSeed' is nonempty and it size is not
//                              consistent with the type of 'parentHandle'; or
//                              unmarshaling sensitive value from 'duplicate' failed
//      TPM_RC_SYMMETRIC        'objectPublic' is either a storage key with no
//                              symmetric algorithm or a non-storage key with
//                              symmetric algorithm different from TPM_ALG_NULL
//      TPM_RC_TYPE             unsupported type of 'objectPublic'; or
//                              'parentHandle' is not a storage key; or
//                              only the public portion of 'parentHandle' is loaded;
//                              or 'objectPublic' and 'duplicate' are of different
//                              types
//      TPM_RC_VALUE            nonempty 'inSymSeed' and its numeric value is
//                              greater than the modulus of the key referenced by
//                              'parentHandle' or 'inSymSeed' is larger than the
//                              size of the digest produced by the name algorithm of
//                              the symmetric key referenced by 'parentHandle'
TPM_RC
TPM2_Import(Import_In*  in,  // IN: input parameter list
            Import_Out* out  // OUT: output parameter list
)
{
    TPM_RC          result = TPM_RC_SUCCESS;
    OBJECT*         parentObject;
    TPM2B_DATA      data;  // symmetric key
    TPMT_SENSITIVE sensitive;
    TPM2B_NAME      name;
    TPMA_OBJECT     attributes;
    UINT16          innerKeySize = 0;  // encrypt key size for inner
                                       // wrapper

    // Input Validation
    // to save typing
    attributes = in->objectPublic.publicArea.objectAttributes;
    // FixedTPM and fixedParent must be CLEAR
    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM)
       || IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent))
        return TPM_RCS_ATTRIBUTES + RC_Import_objectPublic;

    // Get parent pointer
    parentObject = HandleToObject(in->parentHandle);
```

```
    if(!ObjectIsParent(parentObject))
        return TPM_RCS_TYPE + RC_Import_parentHandle;

    if(in->symmetricAlg.algorithm != TPM_ALG_NULL)
    {
        // Get inner wrap key size
        innerKeySize = in->symmetricAlg.keyBits.sym;
        // Input symmetric key must match the size of algorithm.
        if(in->encryptionKey.t.size != (innerKeySize + 7) / 8)
            return TPM_RCS_SIZE + RC_Import_encryptionKey;
    }
    else
    {
        // If input symmetric algorithm is NULL, input symmetric key size must
        // be 0 as well
        if(in->encryptionKey.t.size != 0)
            return TPM_RCS_SIZE + RC_Import_encryptionKey;
        // If encryptedDuplication is SET, then the object must have an inner
        // wrapper
        if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, encryptedDuplication))
            return TPM_RCS_ATTRIBUTES + RC_Import_encryptionKey;
    }
    // See if there is an outer wrapper
    if(in->inSymSeed.t.size != 0)
    {
        // in->inParentHandle is a parent, but in order to decrypt an outer wrapper,
        // it must be able to do key exchange and a symmetric key can't do that.
        if(parentObject->publicArea.type == TPM_ALG_SYMCIPHER)
            return TPM_RCS_TYPE + RC_Import_parentHandle;

        // Decrypt input secret data via asymmetric decryption. TPM_RC_ATTRIBUTES,
        // TPM_RC_ECC_POINT, TPM_RC_INSUFFICIENT, TPM_RC_KEY, TPM_RC_NO_RESULT,
        // TPM_RC_SIZE, TPM_RC_VALUE may be returned at this point
        result = CryptSecretDecrypt(
            parentObject, NULL, DUPLICATE_STRING, &in->inSymSeed, &data);
        pAssert(result != TPM_RC_BINDING);
        if(result != TPM_RC_SUCCESS)
            return RcSafeAddToResult(result, RC_Import_inSymSeed);
    }
    else
    {
        // If encrytpedDuplication is set, then the object must have an outer
        // wrapper
        if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, encryptedDuplication))
            return TPM_RCS_ATTRIBUTES + RC_Import_inSymSeed;
        data.t.size = 0;
    }
    // Compute name of object
    PublicMarshalAndComputeName(&(in->objectPublic.publicArea), &name);
    if(name.t.size == 0)
        return TPM_RCS_HASH + RC_Import_objectPublic;

    // Retrieve sensitive from private.
    // TPM_RC_INSUFFICIENT, TPM_RC_INTEGRITY, TPM_RC_SIZE may be returned here.
    result = DuplicateToSensitive(&in->duplicate.b,
                                  &name.b,
                                  parentObject,
                                  in->objectPublic.publicArea.nameAlg,
                                  &data.b,
                                  &in->symmetricAlg,
                                  &in->encryptionKey.b,
                                  &sensitive);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_Import_duplicate);
```

```
    // If the parent of this object has fixedTPM SET, then validate this
    // object as if it were being loaded so that validation can be skipped
    // when it is actually loaded.
    if(IS_ATTRIBUTE(parentObject->publicArea.objectAttributes, TPMA_OBJECT, fixedTPM))
    {
        result = ObjectLoad(NULL,
                            NULL,
                            &in->objectPublic.publicArea,
                            &sensitive,
                            RC_Import_objectPublic,
                            RC_Import_duplicate,
                            NULL);
    }
    // Command output
    if(result == TPM_RC_SUCCESS)
    {
        // Prepare output private data from sensitive
        SensitiveToPrivate(&sensitive,
                           &name,
                           parentObject,
                           in->objectPublic.publicArea.nameAlg,
                           &out->outPrivate);
    }
    return result;
}

#endif  // CC_Import
```

7.40 /tpm/src/command/Duplication/Rewrap.c

```
#include "Tpm.h"
#include "Rewrap_fp.h"

#if CC_Rewrap  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// This command allows the TPM to serve in the role as an MA.
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'newParent' is not a decryption key
//      TPM_RC_HANDLE           'oldParent' is not consistent with inSymSeed
//      TPM_RC_INTEGRITY        the integrity check of 'inDuplicate' failed
//      TPM_RC_KEY              for an ECC key, the public key is not on the curve
//                              of the curve ID
//      TPM_RC_KEY_SIZE         the decrypted input symmetric key size
//                              does not match the symmetric algorithm
//                              key size of 'oldParent'
//      TPM_RC_TYPE             'oldParent' is not a storage key, or 'newParent'
//                              is not a storage key
//      TPM_RC_VALUE            for an 'oldParent'; RSA key, the data to be decrypted
//                              is greater than the public exponent
//      Unmarshal errors        errors during unmarshaling the input
//                              encrypted buffer to a ECC public key, or
//                              unmarshal the private buffer to 'sensitive'
TPM_RC
TPM2_Rewrap(Rewrap_In*  in,  // IN: input parameter list
            Rewrap_Out* out  // OUT: output parameter list
)
{
    TPM_RC        result = TPM_RC_SUCCESS;
    TPM2B_DATA    data;  // symmetric key
    UINT16        hashSize = 0;
    TPM2B_PRIVATE privateBlob;  // A temporary private blob
```

```
                                // to transit between old
                                // and new wrappers
                                // Input Validation
    if((in->inSymSeed.t.size == 0 && in->oldParent != TPM_RH_NULL)
       || (in->inSymSeed.t.size != 0 && in->oldParent == TPM_RH_NULL))
        return TPM_RCS_HANDLE + RC_Rewrap_oldParent;
    if(in->oldParent != TPM_RH_NULL)
    {
        OBJECT* oldParent = HandleToObject(in->oldParent);

        // old parent key must be a storage object
        if(!ObjectIsStorage(in->oldParent))
            return TPM_RCS_TYPE + RC_Rewrap_oldParent;
        // Decrypt input secret data via asymmetric decryption.  A
        // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
        // point
        result = CryptSecretDecrypt(
            oldParent, NULL, DUPLICATE_STRING, &in->inSymSeed, &data);
        if(result != TPM_RC_SUCCESS)
            return TPM_RCS_VALUE + RC_Rewrap_inSymSeed;
        // Unwrap Outer
        result = UnwrapOuter(oldParent,
                             &in->name.b,
                             oldParent->publicArea.nameAlg,
                             &data.b,
                             FALSE,
                             in->inDuplicate.t.size,
                             in->inDuplicate.t.buffer);
        if(result != TPM_RC_SUCCESS)
            return RcSafeAddToResult(result, RC_Rewrap_inDuplicate);
        // Copy unwrapped data to temporary variable, remove the integrity field
        hashSize =
            sizeof(UINT16) + CryptHashGetDigestSize(oldParent->publicArea.nameAlg);
        privateBlob.t.size = in->inDuplicate.t.size - hashSize;
        pAssert(privateBlob.t.size <= sizeof(privateBlob.t.buffer));
        MemoryCopy(privateBlob.t.buffer,
                   in->inDuplicate.t.buffer + hashSize,
                   privateBlob.t.size);
    }
    else
    {
        // No outer wrap from input blob.  Direct copy.
        privateBlob = in->inDuplicate;
    }
    if(in->newParent != TPM_RH_NULL)
    {
        OBJECT* newParent;
        newParent = HandleToObject(in->newParent);

        // New parent must be a storage object
        if(!ObjectIsStorage(in->newParent))
            return TPM_RCS_TYPE + RC_Rewrap_newParent;
        // Make new encrypt key and its associated secret structure.  A
        // TPM_RC_VALUE error may be returned at this point if RSA algorithm is
        // enabled in TPM
        out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
        result =
            CryptSecretEncrypt(newParent, DUPLICATE_STRING, &data, &out->outSymSeed);
        if(result != TPM_RC_SUCCESS)
            return result;
        // Copy temporary variable to output, reserve the space for integrity
        hashSize =
            sizeof(UINT16) + CryptHashGetDigestSize(newParent->publicArea.nameAlg);
        // Make sure that everything fits into the output buffer
        // Note: this is mostly only an issue if there was no outer wrapper on
        // 'inDuplicate'. It could be as large as a TPM2B_PRIVATE buffer. If we add
```

```
        // a digest for an outer wrapper, it won't fit anymore.
        if((privateBlob.t.size + hashSize) > sizeof(out->outDuplicate.t.buffer))
            return TPM_RCS_VALUE + RC_Rewrap_inDuplicate;
        // Command output
        out->outDuplicate.t.size = privateBlob.t.size;
        pAssert(privateBlob.t.size <= sizeof(out->outDuplicate.t.buffer) - hashSize);
        MemoryCopy(out->outDuplicate.t.buffer + hashSize,
                   privateBlob.t.buffer,
                   privateBlob.t.size);
        // Produce outer wrapper for output
        out->outDuplicate.t.size = ProduceOuterWrap(newParent,
                                                    &in->name.b,
                                                    newParent->publicArea.nameAlg,
                                                    &data.b,
                                                    FALSE,
                                                    out->outDuplicate.t.size,
                                                    out->outDuplicate.t.buffer);
    }
    else  // New parent is a null key so there is no seed
    {
        out->outSymSeed.t.size = 0;

        // Copy privateBlob directly
        out->outDuplicate = privateBlob;
    }
    return TPM_RC_SUCCESS;
}

#endif  // CC_Rewrap
```

7.41 /tpm/src/command/EA/PolicyAuthorize.c

```
#include "Tpm.h"
#include "PolicyAuthorize_fp.h"

#if CC_PolicyAuthorize  // Conditional expansion of this file

#  include "Policy_spt_fp.h"

/*(See part 3 specification)
// Change policy by a signature from authority
*/
//  Return Type: TPM_RC
//      TPM_RC_HASH         hash algorithm in 'keyName' is not supported
//      TPM_RC_SIZE         'keyName' is not the correct size for its hash algorithm
//      TPM_RC_VALUE        the current policyDigest of 'policySession' does not
//                          match 'approvedPolicy'; or 'checkTicket' doesn't match
//                          the provided values
TPM_RC
TPM2_PolicyAuthorize(PolicyAuthorize_In* in  // IN: input parameter list
)
{
    TPM_RC            result = TPM_RC_SUCCESS;
    SESSION*          session;
    TPM2B_DIGEST      authHash;
    HASH_STATE        hashState;
    TPMT_TK_VERIFIED ticket;
    TPM_ALG_ID        hashAlg;
    UINT16            digestSize;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);
```

```
    if(in->keySign.t.size < 2)
    {
        return TPM_RCS_SIZE + RC_PolicyAuthorize_keySign;
    }

    // Extract from the Name of the key, the algorithm used to compute its Name
    hashAlg = BYTE_ARRAY_TO_UINT16(in->keySign.t.name);

    // 'keySign' parameter needs to use a supported hash algorithm, otherwise
    // can't tell how large the digest should be
    if(!CryptHashIsValidAlg(hashAlg, FALSE))
        return TPM_RCS_HASH + RC_PolicyAuthorize_keySign;

    digestSize = CryptHashGetDigestSize(hashAlg);
    if(digestSize != (in->keySign.t.size - 2))
        return TPM_RCS_SIZE + RC_PolicyAuthorize_keySign;

    //If this is a trial policy, skip all validations
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // Check that "approvedPolicy" matches the current value of the
        // policyDigest in policy session
        if(!MemoryEqual2B(&session->u2.policyDigest.b, &in->approvedPolicy.b))
            return TPM_RCS_VALUE + RC_PolicyAuthorize_approvedPolicy;

        // Validate ticket TPMT_TK_VERIFIED
        // Compute aHash.  The authorizing object sign a digest
        //  aHash := hash(approvedPolicy || policyRef).
        // Start hash
        authHash.t.size = CryptHashStart(&hashState, hashAlg);

        // add approvedPolicy
        CryptDigestUpdate2B(&hashState, &in->approvedPolicy.b);

        // add policyRef
        CryptDigestUpdate2B(&hashState, &in->policyRef.b);

        // complete hash
        CryptHashEnd2B(&hashState, &authHash.b);

        // re-compute TPMT_TK_VERIFIED
        result = TicketComputeVerified(
            in->checkTicket.hierarchy, &authHash, &in->keySign, &ticket);
        if(result != TPM_RC_SUCCESS)
            return result;

        // Compare ticket digest.  If not match, return error
        if(!MemoryEqual2B(&in->checkTicket.digest.b, &ticket.digest.b))
            return TPM_RCS_VALUE + RC_PolicyAuthorize_checkTicket;
    }

    // Internal Data Update

    // Set policyDigest to zero digest
    PolicyDigestClear(session);

    // Update policyDigest
    PolicyContextUpdate(
        TPM_CC_PolicyAuthorize, &in->keySign, &in->policyRef, NULL, 0, session);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyAuthorize
```

7.42 /tpm/src/command/EA/PolicyAuthorizeNV.c

```
#include "Tpm.h"

#if CC_PolicyAuthorizeNV  // Conditional expansion of this file

#  include "PolicyAuthorizeNV_fp.h"
#  include "Policy_spt_fp.h"
#  include "Marshal.h"

/*(See part 3 specification)
// Change policy by a signature from authority
*/
//  Return Type: TPM_RC
//      TPM_RC_HASH         hash algorithm in 'keyName' is not supported or is not
//                          the same as the hash algorithm of the policy session
//      TPM_RC_SIZE         'keyName' is not the correct size for its hash algorithm
//      TPM_RC_VALUE        the current policyDigest of 'policySession' does not
//                          match 'approvedPolicy'; or 'checkTicket' doesn't match
//                          the provided values
TPM_RC
TPM2_PolicyAuthorizeNV(PolicyAuthorizeNV_In* in)
{
    SESSION*    session;
    TPM_RC      result;
    NV_REF      locator;
    NV_INDEX*   nvIndex = NvGetIndexInfo(in->nvIndex, &locator);
    TPM2B_NAME  name;
    TPMT_HA     policyInNv;
    BYTE        nvTemp[sizeof(TPMT_HA)];
    BYTE*       buffer = nvTemp;
    INT32       size;

    // Input Validation
    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Skip checks if this is a trial policy
    if(!session->attributes.isTrialPolicy)
    {
        // Check the authorizations for reading
        // Common read access checks. NvReadAccessChecks() returns
        // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
        // error may be returned at this point
        result = NvReadAccessChecks(
            in->authHandle, in->nvIndex, nvIndex->publicArea.attributes);
        if(result != TPM_RC_SUCCESS)
            return result;

        // Read the contents of the index into a temp buffer
        size = MIN(nvIndex->publicArea.dataSize, sizeof(TPMT_HA));
        NvGetIndexData(nvIndex, locator, 0, (UINT16)size, nvTemp);

        // Unmarshal the contents of the buffer into the internal format of a
        // TPMT_HA so that the hash and digest elements can be accessed from the
        // structure rather than the byte array that is in the Index (written by
        // user of the Index).
        result = TPMT_HA_Unmarshal(&policyInNv, &buffer, &size, FALSE);
        if(result != TPM_RC_SUCCESS)
            return result;

        // Verify that the hash is the same
        if(policyInNv.hashAlg != session->authHashAlg)
            return TPM_RC_HASH;

        // See if the contents of the digest in the Index matches the value
```

```
        // in the policy
        if(!MemoryEqual(&policyInNv.digest,
                        &session->u2.policyDigest.t.buffer,
                        session->u2.policyDigest.t.size))
            return TPM_RC_VALUE;
    }

    // Internal Data Update

    // Set policyDigest to zero digest
    PolicyDigestClear(session);

    // Update policyDigest
    PolicyContextUpdate(TPM_CC_PolicyAuthorizeNV,
                        EntityGetName(in->nvIndex, &name),
                        NULL,
                        NULL,
                        0,
                        session);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyAuthorize
```

7.43 /tpm/src/command/EA/PolicyAuthValue.c

```
#include "Tpm.h"
#include "PolicyAuthValue_fp.h"

#if CC_PolicyAuthValue  // Conditional expansion of this file

#  include "Policy_spt_fp.h"

/*(See part 3 specification)
// allows a policy to be bound to the authorization value of the authorized
// object
*/
TPM_RC
TPM2_PolicyAuthValue(PolicyAuthValue_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyAuthValue;
    HASH_STATE hashState;

    // Internal Data Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
    //   Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  complete the hash and get the results
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);
```

```
    // update isAuthValueNeeded bit in the session context
    session->attributes.isAuthValueNeeded = SET;
    session->attributes.isPasswordNeeded  = CLEAR;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyAuthValue
```

7.44 /tpm/src/command/EA/PolicyCapability.c

```
#include "Tpm.h"
#include "PolicyCapability_fp.h"
#include "Policy_spt_fp.h"
#include "ACT_spt_fp.h"
#include "AlgorithmCap_fp.h"
#include "CommandAudit_fp.h"
#include "CommandCodeAttributes_fp.h"
#include "CryptEccMain_fp.h"
#include "Handle_fp.h"
#include "NvDynamic_fp.h"
#include "Object_fp.h"
#include "PCR_fp.h"
#include "PP_fp.h"
#include "PropertyCap_fp.h"
#include "Session_fp.h"

#if CC_PolicyCapability  // Conditional expansion of this file

/*(See part 3 specification)
// This command performs an immediate policy assertion against the current
// value of a TPM Capability.
*/
//  Return Type: TPM_RC
//      TPM_RC_HANDLE       value of 'property' is in an unsupported handle range
//                          for the TPM_CAP_HANDLES 'capability' value
//      TPM_RC_VALUE        invalid 'capability'; or 'property' is not 0 for the
//                          TPM_CAP_PCRS 'capability' value
//      TPM_RC_SIZE         'operandB' is larger than the size of the capability
//                          data minus 'offset'.
TPM_RC
TPM2_PolicyCapability(PolicyCapability_In* in  // IN: input parameter list
)
{
    union
    {
        TPMS_ALG_PROPERTY      alg;
        TPM_HANDLE             handle;
        TPMA_CC                commandAttributes;
        TPM_CC                 command;
        TPMS_TAGGED_PCR_SELECT pcrSelect;
        TPMS_TAGGED_PROPERTY   tpmProperty;
#  if ALG_ECC
        TPM_ECC_CURVE curve;
#  endif  // ALG_ECC
        TPMS_TAGGED_POLICY policy;
#  if ACT_SUPPORT
        TPMS_ACT_DATA act;
#  endif  // ACT_SUPPORT
    } propertyUnion;

    SESSION*     session;
    BYTE         propertyData[sizeof(propertyUnion)];
    UINT16       propertySize = 0;
    BYTE*        buffer       = propertyData;
```

```
    INT32        bufferSize    = sizeof(propertyData);
    TPM_CC       commandCode   = TPM_CC_PolicyCapability;
    HASH_STATE   hashState;
    TPM2B_DIGEST argHash;

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    if(session->attributes.isTrialPolicy == CLEAR)
    {
        switch(in->capability)
        {
            case TPM_CAP_ALGS:
                if(AlgorithmCapGetOneImplemented((TPM_ALG_ID)in->property,
                                                 &propertyUnion.alg))
                {
                    propertySize = TPMS_ALG_PROPERTY_Marshal(
                        &propertyUnion.alg, &buffer, &bufferSize);
                }
                break;
            case TPM_CAP_HANDLES:
            {
                BOOL foundHandle = FALSE;
                switch(HandleGetType((TPM_HANDLE)in->property))
                {
                    case TPM_HT_TRANSIENT:
                        foundHandle = ObjectCapGetOneLoaded((TPM_HANDLE)in->property);
                        break;
                    case TPM_HT_PERSISTENT:
                        foundHandle = NvCapGetOnePersistent((TPM_HANDLE)in->property);
                        break;
                    case TPM_HT_NV_INDEX:
                        foundHandle = NvCapGetOneIndex((TPM_HANDLE)in->property);
                        break;
                    case TPM_HT_LOADED_SESSION:
                        foundHandle =
                            SessionCapGetOneLoaded((TPM_HANDLE)in->property);
                        break;
                    case TPM_HT_SAVED_SESSION:
                        foundHandle = SessionCapGetOneSaved((TPM_HANDLE)in->property);
                        break;
                    case TPM_HT_PCR:
                        foundHandle = PCRCapGetOneHandle((TPM_HANDLE)in->property);
                        break;
                    case TPM_HT_PERMANENT:
                        foundHandle =
                            PermanentCapGetOneHandle((TPM_HANDLE)in->property);
                        break;
                    default:
                        // Unsupported input handle type
                        return TPM_RCS_HANDLE + RC_PolicyCapability_property;
                        break;
                }
                if(foundHandle)
                {
                    TPM_HANDLE handle = (TPM_HANDLE)in->property;
                    propertySize = TPM_HANDLE_Marshal(&handle, &buffer, &bufferSize);
                }
                break;
            }
            case TPM_CAP_COMMANDS:
                if(CommandCapGetOneCC((TPM_CC)in->property,
                                      &propertyUnion.commandAttributes))
                {
                    propertySize = TPMA_CC_Marshal(
                        &propertyUnion.commandAttributes, &buffer, &bufferSize);
```

```
            }
            break;
        case TPM_CAP_PP_COMMANDS:
            if(PhysicalPresenceCapGetOneCC((TPM_CC)in->property))
            {
                TPM_CC cc    = (TPM_CC)in->property;
                propertySize = TPM_CC_Marshal(&cc, &buffer, &bufferSize);
            }
            break;
        case TPM_CAP_AUDIT_COMMANDS:
            if(CommandAuditCapGetOneCC((TPM_CC)in->property))
            {
                TPM_CC cc    = (TPM_CC)in->property;
                propertySize = TPM_CC_Marshal(&cc, &buffer, &bufferSize);
            }
            break;
        // NOTE: TPM_CAP_PCRS can't work for PolicyCapability since CAP_PCRS
        // requires property to be 0 and always returns all the PCR banks.
        case TPM_CAP_PCR_PROPERTIES:
            if(PCRGetProperty((TPM_PT_PCR)in->property, &propertyUnion.pcrSelect))
            {
                propertySize = TPMS_TAGGED_PCR_SELECT_Marshal(
                    &propertyUnion.pcrSelect, &buffer, &bufferSize);
            }
            break;
        case TPM_CAP_TPM_PROPERTIES:
            if(TPMCapGetOneProperty((TPM_PT)in->property,
                                    &propertyUnion.tpmProperty))
            {
                propertySize = TPMS_TAGGED_PROPERTY_Marshal(
                    &propertyUnion.tpmProperty, &buffer, &bufferSize);
            }
            break;
#   if ALG_ECC
        case TPM_CAP_ECC_CURVES:
        {
            TPM_ECC_CURVE curve = (TPM_ECC_CURVE)in->property;
            if(CryptCapGetOneECCCurve(curve))
            {
                propertySize =
                    TPM_ECC_CURVE_Marshal(&curve, &buffer, &bufferSize);
            }
            break;
        }
#   endif  // ALG_ECC
        case TPM_CAP_AUTH_POLICIES:
            if(HandleGetType((TPM_HANDLE)in->property) != TPM_HT_PERMANENT)
                return TPM_RCS_VALUE + RC_PolicyCapability_property;
            if(PermanentHandleGetOnePolicy((TPM_HANDLE)in->property,
                                           &propertyUnion.policy))
            {
                propertySize = TPMS_TAGGED_POLICY_Marshal(
                    &propertyUnion.policy, &buffer, &bufferSize);
            }
            break;
#   if ACT_SUPPORT
        case TPM_CAP_ACT:
            if(((TPM_RH)in->property < TPM_RH_ACT_0)
               || ((TPM_RH)in->property > TPM_RH_ACT_F))
                return TPM_RCS_VALUE + RC_PolicyCapability_property;
            if(ActGetOneCapability((TPM_HANDLE)in->property, &propertyUnion.act))
            {
                propertySize = TPMS_ACT_DATA_Marshal(
                    &propertyUnion.act, &buffer, &bufferSize);
            }
            break;
```

```
#  endif  // ACT_SUPPORT
            case TPM_CAP_VENDOR_PROPERTY:
                // vendor property is not implemented
            default:
                // Unsupported TPM_CAP value
                return TPM_RCS_VALUE + RC_PolicyCapability_capability;
                break;
        }

        if(propertySize == 0)
        {
            // A property that doesn't exist trivially satisfies NEQ, and
            // trivially can't satisfy any other operation.
            if(in->operation != TPM_EO_NEQ)
            {
                return TPM_RC_POLICY;
            }
        }
        else
        {
            // The property was found, so we need to perform the comparison.

            // Make sure that offset is within range
            if(in->offset > propertySize)
            {
                return TPM_RCS_VALUE + RC_PolicyCapability_offset;
            }

            // Property data size should not be smaller than input operandB size
            if((propertySize - in->offset) < in->operandB.t.size)
            {
                return TPM_RCS_SIZE + RC_PolicyCapability_operandB;
            }

            if(!PolicySptCheckCondition(in->operation,
                                        propertyData + in->offset,
                                        in->operandB.t.buffer,
                                        in->operandB.t.size))
            {
                return TPM_RC_POLICY;
            }
        }
    }
    // Internal Data Update

    // Start argument hash
    argHash.t.size = CryptHashStart(&hashState, session->authHashAlg);

    //  add operandB
    CryptDigestUpdate2B(&hashState, &in->operandB.b);

    //  add offset
    CryptDigestUpdateInt(&hashState, sizeof(UINT16), in->offset);

    //  add operation
    CryptDigestUpdateInt(&hashState, sizeof(TPM_EO), in->operation);

    //  add capability
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CAP), in->capability);

    //  add property
    CryptDigestUpdateInt(&hashState, sizeof(UINT32), in->property);

    //  complete argument digest
    CryptHashEnd2B(&hashState, &argHash.b);
```

```
    // Update policyDigest
    //  Start digest
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add argument digest
    CryptDigestUpdate2B(&hashState, &argHash.b);

    // complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyCapability
```

7.45 /tpm/src/command/EA/PolicyCommandCode.c

```
#include "Tpm.h"
#include "PolicyCommandCode_fp.h"

#if CC_PolicyCommandCode  // Conditional expansion of this file

/*(See part 3 specification)
// Add a Command Code restriction to the policyDigest
*/
//  Return Type: TPM_RC
//      TPM_RC_VALUE        'commandCode' of 'policySession' previously set to
//                          a different value

TPM_RC
TPM2_PolicyCommandCode(PolicyCommandCode_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyCommandCode;
    HASH_STATE hashState;

    // Input validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    if(session->commandCode != 0 && session->commandCode != in->code)
        return TPM_RCS_VALUE + RC_PolicyCommandCode_code;
    if(CommandCodeToCommandIndex(in->code) == UNIMPLEMENTED_COMMAND_INDEX)
        return TPM_RCS_POLICY_CC + RC_PolicyCommandCode_code;

    // Internal Data Update
    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCommandCode || code)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);
```

```
    //  add input commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), in->code);

    //  complete the hash and get the results
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // update commandCode value in session context
    session->commandCode = in->code;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyCommandCode
```

7.46 /tpm/src/command/EA/PolicyCounterTimer.c

```
#include "Tpm.h"
#include "PolicyCounterTimer_fp.h"

#if CC_PolicyCounterTimer  // Conditional expansion of this file

#  include "Policy_spt_fp.h"

/*(See part 3 specification)
// Add a conditional gating of a policy based on the contents of the
// TPMS_TIME_INFO structure.
*/
//  Return Type: TPM_RC
//      TPM_RC_POLICY           the comparison of the selected portion of the
//                              TPMS_TIME_INFO with 'operandB' failed
//      TPM_RC_RANGE            'offset' + 'size' exceed size of TPMS_TIME_INFO
//                              structure
TPM_RC
TPM2_PolicyCounterTimer(PolicyCounterTimer_In* in  // IN: input parameter list
)
{
    SESSION*     session;
    TIME_INFO    infoData;  // data buffer of  TPMS_TIME_INFO
    BYTE*        pInfoData = (BYTE*)&infoData;
    UINT16       infoDataSize;
    TPM_CC       commandCode = TPM_CC_PolicyCounterTimer;
    HASH_STATE   hashState;
    TPM2B_DIGEST argHash;

    // Input Validation
    // Get a marshaled time structure
    infoDataSize = TimeGetMarshaled(&infoData);
    // Make sure that the referenced stays within the bounds of the structure.
    // NOTE: the offset checks are made even for a trial policy because the policy
    // will not make any sense if the references are out of bounds of the timer
    // structure.
    if(in->offset > infoDataSize)
        return TPM_RCS_VALUE + RC_PolicyCounterTimer_offset;
    if((UINT32)in->offset + (UINT32)in->operandB.t.size > infoDataSize)
        return TPM_RCS_RANGE;
    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    //If this is a trial policy, skip the check to see if the condition is met.
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // If the command is going to use any part of the counter or timer, need
        // to verify that time is advancing.
        // The time and clock vales are the first two 64-bit values in the clock
        if(in->offset < sizeof(UINT64) + sizeof(UINT64))
```

```
        {
            // Using Clock or Time so see if clock is running. Clock doesn't
            // run while NV is unavailable.
            // TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned here.
            RETURN_IF_NV_IS_NOT_AVAILABLE;
        }
        // offset to the starting position
        pInfoData = (BYTE*)infoData;
        // Check to see if the condition is valid
        if(!PolicySptCheckCondition(in->operation,
                                    pInfoData + in->offset,
                                    in->operandB.t.buffer,
                                    in->operandB.t.size))
            return TPM_RC_POLICY;
    }
    // Internal Data Update
    // Start argument list hash
    argHash.t.size = CryptHashStart(&hashState, session->authHashAlg);
    //  add operandB
    CryptDigestUpdate2B(&hashState, &in->operandB.b);
    //  add offset
    CryptDigestUpdateInt(&hashState, sizeof(UINT16), in->offset);
    //  add operation
    CryptDigestUpdateInt(&hashState, sizeof(TPM_EO), in->operation);
    //  complete argument hash
    CryptHashEnd2B(&hashState, &argHash.b);

    // update policyDigest
    //  start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add argument digest
    CryptDigestUpdate2B(&hashState, &argHash.b);

    // complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyCounterTimer
```

7.47 /tpm/src/command/EA/PolicyCpHash.c

```
#include "Tpm.h"
#include "PolicyCpHash_fp.h"

#if CC_PolicyCpHash  // Conditional expansion of this file

/*(See part 3 specification)
// Add a cpHash restriction to the policyDigest
*/
//  Return Type: TPM_RC
//      TPM_RC_CPHASH           cpHash of 'policySession' has previously been set
//                              to a different value
//      TPM_RC_SIZE             'cpHashA' is not the size of a digest produced
//                              by the hash algorithm associated with
//                              'policySession'
TPM_RC
```

```
TPM2_PolicyCpHash(PolicyCpHash_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyCpHash;
    HASH_STATE hashState;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // A valid cpHash must have the same size as session hash digest
    // NOTE: the size of the digest can't be zero because TPM_ALG_NULL
    // can't be used for the authHashAlg.
    if(in->cpHashA.t.size != CryptHashGetDigestSize(session->authHashAlg))
        return TPM_RCS_SIZE + RC_PolicyCpHash_cpHashA;

    // error if the cpHash in session context is not empty and is not the same
    // as the input or is not a cpHash
    if((IsCpHashUnionOccupied(session->attributes))
       && (!session->attributes.isCpHashDefined
           || !MemoryEqual2B(&in->cpHashA.b, &session->u1.cpHash.b)))
        return TPM_RC_CPHASH;

    // Internal Data Update

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCpHash || cpHashA)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add cpHashA
    CryptDigestUpdate2B(&hashState, &in->cpHashA.b);

    //  complete the digest and get the results
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // update cpHash in session context
    session->u1.cpHash                  = in->cpHashA;
    session->attributes.isCpHashDefined = SET;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyCpHash
```

7.48 /tpm/src/command/EA/PolicyDuplicationSelect.c

```
#include "Tpm.h"
#include "PolicyDuplicationSelect_fp.h"

#if CC_PolicyDuplicationSelect  // Conditional expansion of this file

/*(See part 3 specification)
// allows qualification of duplication so that it a specific new parent may be
// selected or a new parent selected for a specific object.
*/
//  Return Type: TPM_RC
```

```
//      TPM_RC_COMMAND_CODE   'commandCode' of 'policySession' is not empty
//      TPM_RC_CPHASH         'nameHash' of 'policySession' is not empty
TPM_RC
TPM2_PolicyDuplicationSelect(
    PolicyDuplicationSelect_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    HASH_STATE hashState;
    TPM_CC     commandCode = TPM_CC_PolicyDuplicationSelect;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // nameHash in session context must be empty
    if(session->u1.nameHash.t.size != 0)
        return TPM_RC_CPHASH;

    // commandCode in session context must be empty
    if(session->commandCode != 0)
        return TPM_RC_COMMAND_CODE;

    // Internal Data Update

    // Update name hash
    session->u1.nameHash.t.size = CryptHashStart(&hashState, session->authHashAlg);

    //  add objectName
    CryptDigestUpdate2B(&hashState, &in->objectName.b);

    //  add new parent name
    CryptDigestUpdate2B(&hashState, &in->newParentName.b);

    //  complete hash
    CryptHashEnd2B(&hashState, &session->u1.nameHash.b);
    session->attributes.isNameHashDefined = SET;

    // update policy hash
    // Old policyDigest size should be the same as the new policyDigest size since
    // they are using the same hash algorithm
    session->u2.policyDigest.t.size =
        CryptHashStart(&hashState, session->authHashAlg);
    //  add old policy
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add command code
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add objectName
    if(in->includeObject == YES)
        CryptDigestUpdate2B(&hashState, &in->objectName.b);

    //  add new parent name
    CryptDigestUpdate2B(&hashState, &in->newParentName.b);

    //  add includeObject
    CryptDigestUpdateInt(&hashState, sizeof(TPMI_YES_NO), in->includeObject);

    //  complete digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // set commandCode in session context
    session->commandCode = TPM_CC_Duplicate;
```

```
    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyDuplicationSelect
```

7.49 /tpm/src/command/EA/PolicyGetDigest.c

```
#include "Tpm.h"
#include "PolicyGetDigest_fp.h"

#if CC_PolicyGetDigest  // Conditional expansion of this file

/*(See part 3 specification)
// returns the current policyDigest of the session
*/
TPM_RC
TPM2_PolicyGetDigest(PolicyGetDigest_In*  in,  // IN: input parameter list
                     PolicyGetDigest_Out* out  // OUT: output parameter list
)
{
    SESSION* session;

    // Command Output

    // Get pointer to the session structure
    session           = SessionGet(in->policySession);

    out->policyDigest = session->u2.policyDigest;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyGetDigest
```

7.50 /tpm/src/command/EA/PolicyLocality.c

```
#include "Tpm.h"
#include "PolicyLocality_fp.h"
#include "Marshal.h"

#if CC_PolicyLocality  // Conditional expansion of this file

//  Return Type: TPM_RC
//      TPM_RC_RANGE          all the locality values selected by
//                            'locality' have been disabled
//                            by previous TPM2_PolicyLocality() calls.
TPM_RC
TPM2_PolicyLocality(PolicyLocality_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    BYTE       marshalBuffer[sizeof(TPMA_LOCALITY)];
    BYTE       prevSetting[sizeof(TPMA_LOCALITY)];
    UINT32     marshalSize;
    BYTE*      buffer;
    TPM_CC     commandCode = TPM_CC_PolicyLocality;
    HASH_STATE hashState;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Get new locality setting in canonical form
```

```
    marshalBuffer[0] = 0;  // Code analysis says that this is not initialized
    buffer           = marshalBuffer;
    marshalSize      = TPMA_LOCALITY_Marshal(&in->locality, &buffer, NULL);

    // Its an error if the locality parameter is zero
    if(marshalBuffer[0] == 0)
        return TPM_RCS_RANGE + RC_PolicyLocality_locality;

    // Get existing locality setting in canonical form
    prevSetting[0] = 0;  // Code analysis says that this is not initialized
    buffer         = prevSetting;
    TPMA_LOCALITY_Marshal(&session->commandLocality, &buffer, NULL);

    // If the locality has previously been set
    if(prevSetting[0] != 0
       // then the current locality setting and the requested have to be the same
       // type (that is, either both normal or both extended
       && ((prevSetting[0] < 32) != (marshalBuffer[0] < 32)))
        return TPM_RCS_RANGE + RC_PolicyLocality_locality;

    // See if the input is a regular or extended locality
    if(marshalBuffer[0] < 32)
    {
        // if there was no previous setting, start with all normal localities
        // enabled
        if(prevSetting[0] == 0)
            prevSetting[0] = 0x1F;

        // AND the new setting with the previous setting and store it in prevSetting
        prevSetting[0] &= marshalBuffer[0];

        // The result setting can not be 0
        if(prevSetting[0] == 0)
            return TPM_RCS_RANGE + RC_PolicyLocality_locality;
    }
    else
    {
        // for extended locality
        // if the locality has already been set, then it must match the
        if(prevSetting[0] != 0 && prevSetting[0] != marshalBuffer[0])
            return TPM_RCS_RANGE + RC_PolicyLocality_locality;

        // Setting is OK
        prevSetting[0] = marshalBuffer[0];
    }

    // Internal Data Update

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyLocality || locality)
    // Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    // add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    // add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    // add input locality
    CryptDigestUpdate(&hashState, marshalSize, marshalBuffer);

    // complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // update session locality by unmarshal function.  The function must succeed
```

```
    // because both input and existing locality setting have been validated.
    buffer = prevSetting;
    TPMA_LOCALITY_Unmarshal(&session->commandLocality, &buffer, (INT32*)&marshalSize);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyLocality
```

7.51 /tpm/src/command/EA/PolicyNameHash.c

```
#include "Tpm.h"
#include "PolicyNameHash_fp.h"

#if CC_PolicyNameHash  // Conditional expansion of this file

/*(See part 3 specification)
// Add a nameHash restriction to the policyDigest
*/
//  Return Type: TPM_RC
//      TPM_RC_CPHASH     'nameHash' has been previously set to a different value
//      TPM_RC_SIZE       'nameHash' is not the size of the digest produced by the
//                        hash algorithm associated with 'policySession'
TPM_RC
TPM2_PolicyNameHash(PolicyNameHash_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyNameHash;
    HASH_STATE hashState;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // A valid nameHash must have the same size as session hash digest
    // Since the authHashAlg for a session cannot be TPM_ALG_NULL, the digest size
    // is always non-zero.
    if(in->nameHash.t.size != CryptHashGetDigestSize(session->authHashAlg))
        return TPM_RCS_SIZE + RC_PolicyNameHash_nameHash;

    // error if the nameHash in session context is not empty
    if(IsCpHashUnionOccupied(session->attributes))
        return TPM_RC_CPHASH;

    // Internal Data Update

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyNameHash || nameHash)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add nameHash
    CryptDigestUpdate2B(&hashState, &in->nameHash.b);

    //  complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);
```

```
    // update nameHash in session context
    session->u1.nameHash                  = in->nameHash;
    session->attributes.isNameHashDefined = SET;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyNameHash
```

7.52 /tpm/src/command/EA/PolicyNV.c

```
#include "Tpm.h"
#include "PolicyNV_fp.h"

#if CC_PolicyNV  // Conditional expansion of this file

#  include "Policy_spt_fp.h"

/*(See part 3 specification)
// Do comparison to NV location
*/
//  Return Type: TPM_RC
//      TPM_RC_AUTH_TYPE            NV index authorization type is not correct
//      TPM_RC_NV_LOCKED            NV index read locked
//      TPM_RC_NV_UNINITIALIZED     the NV index has not been initialized
//      TPM_RC_POLICY               the comparison to the NV contents failed
//      TPM_RC_SIZE                 the size of 'nvIndex' data starting at 'offset'
//                                  is less than the size of 'operandB'
//      TPM_RC_VALUE                'offset' is too large
TPM_RC
TPM2_PolicyNV(PolicyNV_In* in  // IN: input parameter list
)
{
    TPM_RC       result;
    SESSION*     session;
    NV_REF       locator;
    NV_INDEX*    nvIndex;
    BYTE         nvBuffer[sizeof(in->operandB.t.buffer)];
    TPM2B_NAME   nvName;
    TPM_CC       commandCode = TPM_CC_PolicyNV;
    HASH_STATE   hashState;
    TPM2B_DIGEST argHash;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    //If this is a trial policy, skip all validations and the operation
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // No need to access the actual NV index information for a trial policy.
        nvIndex = NvGetIndexInfo(in->nvIndex, &locator);

        // Common read access checks. NvReadAccessChecks() may return
        // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
        result = NvReadAccessChecks(
            in->authHandle, in->nvIndex, nvIndex->publicArea.attributes);
        if(result != TPM_RC_SUCCESS)
            return result;

        // Make sure that offset is within range
        if(in->offset > nvIndex->publicArea.dataSize)
            return TPM_RCS_VALUE + RC_PolicyNV_offset;
```

```
        // Valid NV data size should not be smaller than input operandB size
        if((nvIndex->publicArea.dataSize - in->offset) < in->operandB.t.size)
            return TPM_RCS_SIZE + RC_PolicyNV_operandB;

        // Get NV data.  The size of NV data equals the input operand B size
        NvGetIndexData(nvIndex, locator, in->offset, in->operandB.t.size, nvBuffer);

        // Check to see if the condition is valid
        if(!PolicySptCheckCondition(
               in->operation, nvBuffer, in->operandB.t.buffer, in->operandB.t.size))
            return TPM_RC_POLICY;
    }
    // Internal Data Update

    // Start argument hash
    argHash.t.size = CryptHashStart(&hashState, session->authHashAlg);

    //  add operandB
    CryptDigestUpdate2B(&hashState, &in->operandB.b);

    //  add offset
    CryptDigestUpdateInt(&hashState, sizeof(UINT16), in->offset);

    //  add operation
    CryptDigestUpdateInt(&hashState, sizeof(TPM_EO), in->operation);

    //  complete argument digest
    CryptHashEnd2B(&hashState, &argHash.b);

    // Update policyDigest
    //  Start digest
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add argument digest
    CryptDigestUpdate2B(&hashState, &argHash.b);

    // Adding nvName
    CryptDigestUpdate2B(&hashState, &EntityGetName(in->nvIndex, &nvName)->b);

    // complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyNV
```

7.53 /tpm/src/command/EA/PolicyNvWritten.c

```
#include "Tpm.h"
#include "PolicyNvWritten_fp.h"

#if CC_PolicyNvWritten  // Conditional expansion of this file

// Make an NV Index policy dependent on the state of the TPMA_NV_WRITTEN
// attribute of the index.
//  Return Type: TPM_RC
//      TPM_RC_VALUE         a conflicting request for the attribute has
//                           already been processed
```

```
TPM_RC
TPM2_PolicyNvWritten(PolicyNvWritten_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyNvWritten;
    HASH_STATE hashState;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // If already set is this a duplicate (the same setting)? If it
    // is a conflicting setting, it is an error
    if(session->attributes.checkNvWritten == SET)
    {
        if(((session->attributes.nvWrittenState == SET) != (in->writtenSet == YES)))
            return TPM_RCS_VALUE + RC_PolicyNvWritten_writtenSet;
    }

    // Internal Data Update

    // Set session attributes so that the NV Index needs to be checked
    session->attributes.checkNvWritten = SET;
    session->attributes.nvWrittenState = (in->writtenSet == YES);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyNvWritten
    //                        || writtenSet)
    // Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    // add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    // add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    // add the byte of writtenState
    CryptDigestUpdateInt(&hashState, sizeof(TPMI_YES_NO), in->writtenSet);

    // complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyNvWritten
```

7.54 /tpm/src/command/EA/PolicyOR.c

```
#include "Tpm.h"
#include "PolicyOR_fp.h"

#if CC_PolicyOR  // Conditional expansion of this file

#  include "Policy_spt_fp.h"

/*(See part 3 specification)
// PolicyOR command
*/
//  Return Type: TPM_RC
//      TPM_RC_VALUE            no digest in 'pHashList' matched the current
//                              value of policyDigest for 'policySession'
```

```
TPM_RC
TPM2_PolicyOR(PolicyOR_In* in  // IN: input parameter list
)
{
    SESSION* session;
    UINT32   i;

    // Input Validation and Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Compare and Update Internal Session policy if match
    for(i = 0; i < in->pHashList.count; i++)
    {
        if(session->attributes.isTrialPolicy == SET
           || (MemoryEqual2B(&session->u2.policyDigest.b,
                             &in->pHashList.digests[i].b)))
        {
            // Found a match
            HASH_STATE hashState;
            TPM_CC     commandCode = TPM_CC_PolicyOR;

            // Start hash
            session->u2.policyDigest.t.size =
                CryptHashStart(&hashState, session->authHashAlg);
            // Set policyDigest to 0 string and add it to hash
            MemorySet(session->u2.policyDigest.t.buffer,
                      0,
                      session->u2.policyDigest.t.size);
            CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

            // add command code
            CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

            // Add each of the hashes in the list
            for(i = 0; i < in->pHashList.count; i++)
            {
                // Extend policyDigest
                CryptDigestUpdate2B(&hashState, &in->pHashList.digests[i].b);
            }
            // Complete digest
            CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

            return TPM_RC_SUCCESS;
        }
    }
    // None of the values in the list matched the current policyDigest
    return TPM_RCS_VALUE + RC_PolicyOR_pHashList;
}

#endif  // CC_PolicyOR
```

7.55 /tpm/src/command/EA/PolicyParameters.c

```
#include "Tpm.h"
#include "PolicyParameters_fp.h"

#if CC_PolicyParameters  // Conditional expansion of this file

/*(See part 3 specification)
// Add a parameters restriction to the policyDigest
*/
//  Return Type: TPM_RC
//      TPM_RC_CPHASH     cpHash of 'policySession' has previously been set
```

```
//                          to a different value
//      TPM_RC_SIZE         'pHash' is not the size of the digest produced by the
//                          hash algorithm associated with 'policySession'
TPM_RC
TPM2_PolicyParameters(PolicyParameters_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyParameters;
    HASH_STATE hashState;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // A valid pHash must have the same size as session hash digest
    // Since the authHashAlg for a session cannot be TPM_ALG_NULL, the digest size
    // is always non-zero.
    if(in->pHash.t.size != CryptHashGetDigestSize(session->authHashAlg))
        return TPM_RCS_SIZE + RC_PolicyParameters_pHash;

    // error if the pHash in session context is not empty
    if(IsCpHashUnionOccupied(session->attributes))
        return TPM_RC_CPHASH;

    // Internal Data Update

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyParameters || pHash)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add pHash
    CryptDigestUpdate2B(&hashState, &in->pHash.b);

    //  complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // update pHash in session context
    session->u1.pHash                              = in->pHash;
    session->attributes.isParametersHashDefined = SET;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyParameters
```

7.56 /tpm/src/command/EA/PolicyPassword.c

```
#include "Tpm.h"
#include "PolicyPassword_fp.h"

#if CC_PolicyPassword  // Conditional expansion of this file

#  include "Policy_spt_fp.h"

/*(See part 3 specification)
// allows a policy to be bound to the authorization value of the authorized
```

```
// object
*/
TPM_RC
TPM2_PolicyPassword(PolicyPassword_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyAuthValue;
    HASH_STATE hashState;

    // Internal Data Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    //  Update isPasswordNeeded bit
    session->attributes.isPasswordNeeded  = SET;
    session->attributes.isAuthValueNeeded = CLEAR;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyPassword
```

7.57 /tpm/src/command/EA/PolicyPCR.c

```
#include "Tpm.h"

#if CC_PolicyPCR  // Conditional expansion of this file

#  include "PolicyPCR_fp.h"
#  include "Marshal.h"

/*(See part 3 specification)
// Add a PCR gate for a policy session
*/
//  Return Type: TPM_RC
//      TPM_RC_VALUE          if provided, 'pcrDigest' does not match the
//                            current PCR settings
//      TPM_RC_PCR_CHANGED    a previous TPM2_PolicyPCR() set
//                            pcrCounter and it has changed
TPM_RC
TPM2_PolicyPCR(PolicyPCR_In* in  // IN: input parameter list
)
{
    SESSION*     session;
    TPM2B_DIGEST pcrDigest;
    BYTE         pcrs[sizeof(TPML_PCR_SELECTION)];
    UINT32       pcrSize;
    BYTE*        buffer;
    TPM_CC       commandCode = TPM_CC_PolicyPCR;
```

```
    HASH_STATE   hashState;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Compute current PCR digest
    PCRComputeCurrentDigest(session->authHashAlg, &in->pcrs, &pcrDigest);

    // Do validation for non trial session
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // Make sure that this is not going to invalidate a previous PCR check
        if(session->pcrCounter != 0 && session->pcrCounter != gr.pcrCounter)
            return TPM_RC_PCR_CHANGED;

        // If the caller specified the PCR digest and it does not
        // match the current PCR settings, return an error..
        if(in->pcrDigest.t.size != 0)
        {
            if(!MemoryEqual2B(&in->pcrDigest.b, &pcrDigest.b))
                return TPM_RCS_VALUE + RC_PolicyPCR_pcrDigest;
        }
    }
    else
    {
        // For trial session, just use the input PCR digest if one provided
        // Note: It can't be too big because it is a TPM2B_DIGEST and the size
        // would have been checked during unmarshaling
        if(in->pcrDigest.t.size != 0)
            pcrDigest = in->pcrDigest;
    }
    // Internal Data Update
    // Update policy hash
    // policyDigestnew = hash(   policyDigestold || TPM_CC_PolicyPCR
    //                       || PCRS || pcrDigest)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add PCRS
    buffer  = pcrs;
    pcrSize = TPML_PCR_SELECTION_Marshal(&in->pcrs, &buffer, NULL);
    CryptDigestUpdate(&hashState, pcrSize, pcrs);

    //  add PCR digest
    CryptDigestUpdate2B(&hashState, &pcrDigest.b);

    //  complete the hash and get the results
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    //  update pcrCounter in session context for non trial session
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        session->pcrCounter = gr.pcrCounter;
    }

    return TPM_RC_SUCCESS;
}
```

```
#endif  // CC_PolicyPCR
```

7.58 /tpm/src/command/EA/PolicyPhysicalPresence.c

```
#include "Tpm.h"
#include "PolicyPhysicalPresence_fp.h"

#if CC_PolicyPhysicalPresence  // Conditional expansion of this file

/*(See part 3 specification)
// indicate that physical presence will need to be asserted at the time the
// authorization is performed
*/
TPM_RC
TPM2_PolicyPhysicalPresence(PolicyPhysicalPresence_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyPhysicalPresence;
    HASH_STATE hashState;

    // Internal Data Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyPhysicalPresence)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  complete the digest
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // update session attribute
    session->attributes.isPPRequired = SET;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyPhysicalPresence
```

7.59 /tpm/src/command/EA/PolicySecret.c

```
#include "Tpm.h"
#include "PolicySecret_fp.h"

#if CC_PolicySecret  // Conditional expansion of this file

#  include "Policy_spt_fp.h"
#  include "NV_spt_fp.h"

/*(See part 3 specification)
// Add a secret-based authorization to the policy evaluation
*/
//  Return Type: TPM_RC
//      TPM_RC_CPHASH           cpHash for policy was previously set to a
//                              value that is not the same as 'cpHashA'
```

```
//      TPM_RC_EXPIRED          'expiration' indicates a time in the past
//      TPM_RC_NONCE            'nonceTPM' does not match the nonce associated
//                              with 'policySession'
//      TPM_RC_SIZE             'cpHashA' is not the size of a digest for the
//                              hash associated with 'policySession'
TPM_RC
TPM2_PolicySecret(PolicySecret_In*  in,  // IN: input parameter list
                  PolicySecret_Out* out  // OUT: output parameter list
)
{
    TPM_RC     result;
    SESSION*   session;
    TPM2B_NAME entityName;
    UINT64     authTimeout = 0;
    // Input Validation
    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    //Only do input validation if this is not a trial policy session
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        authTimeout = ComputeAuthTimeout(session, in->expiration, &in->nonceTPM);

        result      = PolicyParameterChecks(session,
                                            authTimeout,
                                            &in->cpHashA,
                                            &in->nonceTPM,
                                            RC_PolicySecret_nonceTPM,
                                            RC_PolicySecret_cpHashA,
                                            RC_PolicySecret_expiration);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    // Internal Data Update
    // Update policy context with input policyRef and name of authorizing key
    // This value is computed even for trial sessions. Possibly update the cpHash
    PolicyContextUpdate(TPM_CC_PolicySecret,
                        EntityGetName(in->authHandle, &entityName),
                        &in->policyRef,
                        &in->cpHashA,
                        authTimeout,
                        session);
    // Command Output
    // Create ticket and timeout buffer if in->expiration < 0 and this is not
    // a trial session.
    // NOTE: PolicyParameterChecks() makes sure that nonceTPM is present
    // when expiration is non-zero.
    if(in->expiration < 0 && session->attributes.isTrialPolicy == CLEAR
       && !NvIsPinPassIndex(in->authHandle))
    {
        BOOL expiresOnReset = (in->nonceTPM.t.size == 0);
        // Compute policy ticket
        authTimeout &= ~EXPIRATION_BIT;
        result = TicketComputeAuth(TPM_ST_AUTH_SECRET,
                                   EntityGetHierarchy(in->authHandle),
                                   authTimeout,
                                   expiresOnReset,
                                   &in->cpHashA,
                                   &in->policyRef,
                                   &entityName,
                                   &out->policyTicket);
        if(result != TPM_RC_SUCCESS)
            return result;

        // Generate timeout buffer.  The format of output timeout buffer is
        // TPM-specific.
```

```
        // Note: In this implementation, the timeout buffer value is computed after
        // the ticket is produced so, when the ticket is checked, the expiration
        // flag needs to be extracted before the ticket is checked.
        out->timeout.t.size = sizeof(authTimeout);
        // In the Windows compatible version, the least-significant bit of the
        // timeout value is used as a flag to indicate if the authorization expires
        // on reset. The flag is the MSb.
        if(expiresOnReset)
            authTimeout |= EXPIRATION_BIT;
        UINT64_TO_BYTE_ARRAY(authTimeout, out->timeout.t.buffer);
    }
    else
    {
        // timeout buffer is null
        out->timeout.t.size = 0;

        // authorization ticket is null
        out->policyTicket.tag           = TPM_ST_AUTH_SECRET;
        out->policyTicket.hierarchy     = TPM_RH_NULL;
        out->policyTicket.digest.t.size = 0;
    }
    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicySecret
```

7.60 /tpm/src/command/EA/PolicySigned.c

```
#include "Tpm.h"
#include "Policy_spt_fp.h"
#include "PolicySigned_fp.h"

#if CC_PolicySigned  // Conditional expansion of this file

/*(See part 3 specification)
// Include an asymmetrically signed authorization to the policy evaluation
*/
//  Return Type: TPM_RC
//      TPM_RC_CPHASH           cpHash was previously set to a different value
//      TPM_RC_EXPIRED          'expiration' indicates a time in the past or
//                              'expiration' is non-zero but no nonceTPM is present
//      TPM_RC_NONCE            'nonceTPM' is not the nonce associated with the
//                              'policySession'
//      TPM_RC_SCHEME           the signing scheme of 'auth' is not supported by the
//                              TPM
//      TPM_RC_SIGNATURE        the signature is not genuine
//      TPM_RC_SIZE             input cpHash has wrong size
TPM_RC
TPM2_PolicySigned(PolicySigned_In*  in,  // IN: input parameter list
                  PolicySigned_Out* out  // OUT: output parameter list
)
{
    TPM_RC       result = TPM_RC_SUCCESS;
    SESSION*     session;
    TPM2B_NAME   entityName;
    TPM2B_DIGEST authHash;
    HASH_STATE   hashState;
    UINT64       authTimeout = 0;
    // Input Validation
    // Set up local pointers
    session = SessionGet(in->policySession);  // the session structure

    // Only do input validation if this is not a trial policy session
    if(session->attributes.isTrialPolicy == CLEAR)
    {
```

```
    authTimeout = ComputeAuthTimeout(session, in->expiration, &in->nonceTPM);

    result      = PolicyParameterChecks(session,
                                        authTimeout,
                                        &in->cpHashA,
                                        &in->nonceTPM,
                                        RC_PolicySigned_nonceTPM,
                                        RC_PolicySigned_cpHashA,
                                        RC_PolicySigned_expiration);
    if(result != TPM_RC_SUCCESS)
        return result;
    // Re-compute the digest being signed
    /*(See part 3 specification)
    // The digest is computed as:
    //     aHash := hash ( nonceTPM | expiration | cpHashA | policyRef)
    //  where:
    //      hash()      the hash associated with the signed authorization
    //      nonceTPM    the nonceTPM value from the TPM2_StartAuthSession .
    //                  response If the authorization is not limited to this
    //                  session, the size of this value is zero.
    //      expiration  time limit on authorization set by authorizing object.
    //                  This 32-bit value is set to zero if the expiration
    //                  time is not being set.
    //      cpHashA     hash of the command parameters for the command being
    //                  approved using the hash algorithm of the PSAP session.
    //                  Set to NULLauth if the authorization is not limited
    //                  to a specific command.
    //      policyRef   hash of an opaque value determined by the authorizing
    //                  object.  Set to the NULLdigest if no hash is present.
    */
    // Start hash
    authHash.t.size = CryptHashStart(&hashState, CryptGetSignHashAlg(&in->auth));
    // If there is no digest size, then we don't have a verification function
    // for this algorithm (e.g. TPM_ALG_ECDAA) so indicate that it is a
    // bad scheme.
    if(authHash.t.size == 0)
        return TPM_RCS_SCHEME + RC_PolicySigned_auth;

    //  nonceTPM
    CryptDigestUpdate2B(&hashState, &in->nonceTPM.b);

    //  expiration
    CryptDigestUpdateInt(&hashState, sizeof(UINT32), in->expiration);

    //  cpHashA
    CryptDigestUpdate2B(&hashState, &in->cpHashA.b);

    //  policyRef
    CryptDigestUpdate2B(&hashState, &in->policyRef.b);

    //  Complete digest
    CryptHashEnd2B(&hashState, &authHash.b);

    // Validate Signature.  A TPM_RC_SCHEME, TPM_RC_HANDLE or TPM_RC_SIGNATURE
    // error may be returned at this point
    result = CryptValidateSignature(in->authObject, &authHash, &in->auth);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_PolicySigned_auth);
}
// Internal Data Update
// Update policy with input policyRef and name of authorization key
// These values are updated even if the session is a trial session
PolicyContextUpdate(TPM_CC_PolicySigned,
                    EntityGetName(in->authObject, &entityName),
                    &in->policyRef,
                    &in->cpHashA,
```

```
                        authTimeout,
                        session);
    // Command Output
    // Create ticket and timeout buffer if in->expiration < 0 and this is not
    // a trial session.
    // NOTE: PolicyParameterChecks() makes sure that nonceTPM is present
    // when expiration is non-zero.
    if(in->expiration < 0 && session->attributes.isTrialPolicy == CLEAR)
    {
        BOOL expiresOnReset = (in->nonceTPM.t.size == 0);
        // Compute policy ticket
        authTimeout &= ~EXPIRATION_BIT;

        result = TicketComputeAuth(TPM_ST_AUTH_SIGNED,
                                   EntityGetHierarchy(in->authObject),
                                   authTimeout,
                                   expiresOnReset,
                                   &in->cpHashA,
                                   &in->policyRef,
                                   &entityName,
                                   &out->policyTicket);
        if(result != TPM_RC_SUCCESS)
            return result;

        // Generate timeout buffer.  The format of output timeout buffer is
        // TPM-specific.
        // Note: In this implementation, the timeout buffer value is computed after
        // the ticket is produced so, when the ticket is checked, the expiration
        // flag needs to be extracted before the ticket is checked.
        // In the Windows compatible version, the least-significant bit of the
        // timeout value is used as a flag to indicate if the authorization expires
        // on reset. The flag is the MSb.
        out->timeout.t.size = sizeof(authTimeout);
        if(expiresOnReset)
            authTimeout |= EXPIRATION_BIT;
        UINT64_TO_BYTE_ARRAY(authTimeout, out->timeout.t.buffer);
    }
    else
    {
        // Generate a null ticket.
        // timeout buffer is null
        out->timeout.t.size = 0;

        // authorization ticket is null
        out->policyTicket.tag           = TPM_ST_AUTH_SIGNED;
        out->policyTicket.hierarchy     = TPM_RH_NULL;
        out->policyTicket.digest.t.size = 0;
    }
    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicySigned
```

7.61 /tpm/src/command/EA/PolicyTemplate.c

```
#include "Tpm.h"
#include "PolicyTemplate_fp.h"

#if CC_PolicyTemplate  // Conditional expansion of this file

/*(See part 3 specification)
// Add a cpHash restriction to the policyDigest
*/
//  Return Type: TPM_RC
//      TPM_RC_CPHASH           cpHash of 'policySession' has previously been set
```

```
//                              to a different value
//      TPM_RC_SIZE             'templateHash' is not the size of a digest produced
//                              by the hash algorithm associated with
//                              'policySession'
TPM_RC
TPM2_PolicyTemplate(PolicyTemplate_In* in  // IN: input parameter list
)
{
    SESSION*   session;
    TPM_CC     commandCode = TPM_CC_PolicyTemplate;
    HASH_STATE hashState;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // error if the templateHash in session context is not empty and is not the
    // same as the input or is not a template
    if((IsCpHashUnionOccupied(session->attributes))
       && (!session->attributes.isTemplateHashDefined
           || !MemoryEqual2B(&in->templateHash.b, &session->u1.templateHash.b)))
        return TPM_RC_CPHASH;

    // A valid templateHash must have the same size as session hash digest
    if(in->templateHash.t.size != CryptHashGetDigestSize(session->authHashAlg))
        return TPM_RCS_SIZE + RC_PolicyTemplate_templateHash;

    // Internal Data Update
    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCpHash
    //  || cpHashA.buffer)
    //  Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    //  add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), commandCode);

    //  add cpHashA
    CryptDigestUpdate2B(&hashState, &in->templateHash.b);

    //  complete the digest and get the results
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // update templateHash in session context
    session->u1.templateHash                  = in->templateHash;
    session->attributes.isTemplateHashDefined = SET;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyTemplateHash
```

7.62 /tpm/src/command/EA/PolicyTicket.c

```
#include "Tpm.h"
#include "PolicyTicket_fp.h"

#if CC_PolicyTicket  // Conditional expansion of this file

#  include "Policy_spt_fp.h"
```

```
/*(See part 3 specification)
// Include ticket to the policy evaluation
*/
//  Return Type: TPM_RC
//      TPM_RC_CPHASH           policy's cpHash was previously set to a different
//                              value
//      TPM_RC_EXPIRED          'timeout' value in the ticket is in the past and the
//                              ticket has expired
//      TPM_RC_SIZE             'timeout' or 'cpHash' has invalid size for the
//      TPM_RC_TICKET           'ticket' is not valid
TPM_RC
TPM2_PolicyTicket(PolicyTicket_In* in  // IN: input parameter list
)
{
    TPM_RC       result;
    SESSION*     session;
    UINT64       authTimeout;
    TPMT_TK_AUTH ticketToCompare;
    TPM_CC       commandCode = TPM_CC_PolicySecret;
    BOOL         expiresOnReset;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // NOTE: A trial policy session is not allowed to use this command.
    // A ticket is used in place of a previously given authorization. Since
    // a trial policy doesn't actually authenticate, the validated
    // ticket is not necessary and, in place of using a ticket, one
    // should use the intended authorization for which the ticket
    // would be a substitute.
    if(session->attributes.isTrialPolicy)
        return TPM_RCS_ATTRIBUTES + RC_PolicyTicket_policySession;
    // Restore timeout data.  The format of timeout buffer is TPM-specific.
    // In this implementation, the most significant bit of the timeout value is
    // used as the flag to indicate that the ticket expires on TPM Reset or
    // TPM Restart. The flag has to be removed before the parameters and ticket
    // are checked.
    if(in->timeout.t.size != sizeof(UINT64))
        return TPM_RCS_SIZE + RC_PolicyTicket_timeout;
    authTimeout = BYTE_ARRAY_TO_UINT64(in->timeout.t.buffer);

    // extract the flag
    expiresOnReset = (authTimeout & EXPIRATION_BIT) != 0;
    authTimeout &= ~EXPIRATION_BIT;

    // Do the normal checks on the cpHashA and timeout values
    result = PolicyParameterChecks(session,
                                   authTimeout,
                                   &in->cpHashA,
                                   NULL,  // no nonce
                                   0,     // no bad nonce return
                                   RC_PolicyTicket_cpHashA,
                                   RC_PolicyTicket_timeout);
    if(result != TPM_RC_SUCCESS)
        return result;
    // Validate Ticket
    // Re-generate policy ticket by input parameters
    result = TicketComputeAuth(in->ticket.tag,
                               in->ticket.hierarchy,
                               authTimeout,
                               expiresOnReset,
                               &in->cpHashA,
                               &in->policyRef,
                               &in->authName,
```

```
                                    &ticketToCompare);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Compare generated digest with input ticket digest
    if(!MemoryEqual2B(&in->ticket.digest.b, &ticketToCompare.digest.b))
        return TPM_RCS_TICKET + RC_PolicyTicket_ticket;

    // Internal Data Update

    // Is this ticket to take the place of a TPM2_PolicySigned() or
    // a TPM2_PolicySecret()?
    if(in->ticket.tag == TPM_ST_AUTH_SIGNED)
        commandCode = TPM_CC_PolicySigned;
    else if(in->ticket.tag == TPM_ST_AUTH_SECRET)
        commandCode = TPM_CC_PolicySecret;
    else
        // There could only be two possible tag values.  Any other value should
        // be caught by the ticket validation process.
        FAIL(FATAL_ERROR_INTERNAL);

    // Update policy context
    PolicyContextUpdate(commandCode,
                        &in->authName,
                        &in->policyRef,
                        &in->cpHashA,
                        authTimeout,
                        session);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyTicket
```

7.63 /tpm/src/command/EA/Policy_spt.c

```
//** Includes
#include "Tpm.h"
#include "Policy_spt_fp.h"
#include "PolicySigned_fp.h"
#include "PolicySecret_fp.h"
#include "PolicyTicket_fp.h"

//** Functions
//*** PolicyParameterChecks()
// This function validates the common parameters of TPM2_PolicySiged()
// and TPM2_PolicySecret(). The common parameters are 'nonceTPM',
// 'expiration', and 'cpHashA'.
TPM_RC
PolicyParameterChecks(SESSION*      session,
                      UINT64        authTimeout,
                      TPM2B_DIGEST* cpHashA,
                      TPM2B_NONCE*  nonce,
                      TPM_RC        blameNonce,
                      TPM_RC        blameCpHash,
                      TPM_RC        blameExpiration)
{
    // Validate that input nonceTPM is correct if present
    if(nonce != NULL && nonce->t.size != 0)
    {
        if(!MemoryEqual2B(&nonce->b, &session->nonceTPM.b))
            return TPM_RCS_NONCE + blameNonce;
    }
    // If authTimeout is set (expiration != 0...
    if(authTimeout != 0)
```

```
    {
        // Validate input expiration.
        // Cannot compare time if clock stop advancing.  A TPM_RC_NV_UNAVAILABLE
        // or TPM_RC_NV_RATE error may be returned here.
        RETURN_IF_NV_IS_NOT_AVAILABLE;

        // if the time has already passed or the time epoch has changed then the
        // time value is no longer good.
        if((authTimeout < g_time) || (session->epoch != g_timeEpoch))
            return TPM_RCS_EXPIRED + blameExpiration;
    }
    // If the cpHash is present, then check it
    if(cpHashA != NULL && cpHashA->t.size != 0)
    {
        // The cpHash input has to have the correct size
        if(cpHashA->t.size != session->u2.policyDigest.t.size)
            return TPM_RCS_SIZE + blameCpHash;

        // If the cpHash has already been set, then this input value
        // must match the current value.
        if(session->u1.cpHash.b.size != 0
           && !MemoryEqual2B(&cpHashA->b, &session->u1.cpHash.b))
            return TPM_RC_CPHASH;
    }
    return TPM_RC_SUCCESS;
}

//*** PolicyContextUpdate()
// Update policy hash
//      Update the policyDigest in policy session by extending policyRef and
//      objectName to it. This will also update the cpHash if it is present.
//
//  Return Type: void
void PolicyContextUpdate(
    TPM_CC        commandCode,    // IN: command code
    TPM2B_NAME*   name,           // IN: name of entity
    TPM2B_NONCE*  ref,            // IN: the reference data
    TPM2B_DIGEST* cpHash,         // IN: the cpHash (optional)
    UINT64        policyTimeout,  // IN: the timeout value for the policy
    SESSION*      session         // IN/OUT: policy session to be updated
)
{
    HASH_STATE hashState;

    // Start hash
    CryptHashStart(&hashState, session->authHashAlg);

    // policyDigest size should always be the digest size of session hash algorithm.
    pAssert(session->u2.policyDigest.t.size
            == CryptHashGetDigestSize(session->authHashAlg));

    // add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

    // add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(commandCode), commandCode);

    // add name if applicable
    if(name != NULL)
        CryptDigestUpdate2B(&hashState, &name->b);

    // Complete the digest and get the results
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);

    // If the policy reference is not null, do a second update to the digest.
    if(ref != NULL)
```

```
    {

        // Start second hash computation
        CryptHashStart(&hashState, session->authHashAlg);

        // add policyDigest
        CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);

        // add policyRef
        CryptDigestUpdate2B(&hashState, &ref->b);

        // Complete second digest
        CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);
    }
    // Deal with the cpHash. If the cpHash value is present
    // then it would have already been checked to make sure that
    // it is compatible with the current value so all we need
    // to do here is copy it and set the isCpHashDefined attribute
    if(cpHash != NULL && cpHash->t.size != 0)
    {
        session->u1.cpHash                   = *cpHash;
        session->attributes.isCpHashDefined = SET;
    }

    // update the timeout if it is specified
    if(policyTimeout != 0)
    {
        // If the timeout has not been set, then set it to the new value
        // than the current timeout then set it to the new value
        if(session->timeout == 0 || session->timeout > policyTimeout)
            session->timeout = policyTimeout;
    }
    return;
}
//*** ComputeAuthTimeout()
// This function is used to determine what the authorization timeout value for
// the session should be.
UINT64
ComputeAuthTimeout(SESSION* session,    // IN: the session containing the time
                                        //     values
                   INT32 expiration,    // IN: either the number of seconds from
                                        //     the start of the session or the
                                        //     time in g_timer;
                   TPM2B_NONCE* nonce   // IN: indicator of the time base
)
{
    UINT64 policyTime;
    // If no expiration, policy time is 0
    if(expiration == 0)
        policyTime = 0;
    else
    {
        if(expiration < 0)
            expiration = -expiration;
        if(nonce->t.size == 0)
            // The input time is absolute Time (not Clock), but it is expressed
            // in seconds. To make sure that we don't time out too early, take the
            // current value of milliseconds in g_time and add that to the input
            // seconds value.
            policyTime = (((UINT64)expiration) * 1000) + g_time % 1000;
        else
            // The policy timeout is the absolute value of the expiration in seconds
            // added to the start time of the policy.
            policyTime = session->startTime + (((UINT64)expiration) * 1000);
    }
    return policyTime;
```

```
}

//*** PolicyDigestClear()
// Function to reset the policyDigest of a session
void PolicyDigestClear(SESSION* session)
{
    session->u2.policyDigest.t.size = CryptHashGetDigestSize(session->authHashAlg);
    MemorySet(session->u2.policyDigest.t.buffer, 0, session->u2.policyDigest.t.size);
}

//*** PolicySptCheckCondition()
// Checks to see if the condition in the policy is satisfied.
BOOL PolicySptCheckCondition(TPM_EO operation, BYTE* opA, BYTE* opB, UINT16 size)
{
    // Arithmetic Comparison
    switch(operation)
    {
        case TPM_EO_EQ:
            // compare A = B
            return (UnsignedCompareB(size, opA, size, opB) == 0);
            break;
        case TPM_EO_NEQ:
            // compare A != B
            return (UnsignedCompareB(size, opA, size, opB) != 0);
            break;
        case TPM_EO_SIGNED_GT:
            // compare A > B signed
            return (SignedCompareB(size, opA, size, opB) > 0);
            break;
        case TPM_EO_UNSIGNED_GT:
            // compare A > B unsigned
            return (UnsignedCompareB(size, opA, size, opB) > 0);
            break;
        case TPM_EO_SIGNED_LT:
            // compare A < B signed
            return (SignedCompareB(size, opA, size, opB) < 0);
            break;
        case TPM_EO_UNSIGNED_LT:
            // compare A < B unsigned
            return (UnsignedCompareB(size, opA, size, opB) < 0);
            break;
        case TPM_EO_SIGNED_GE:
            // compare A >= B signed
            return (SignedCompareB(size, opA, size, opB) >= 0);
            break;
        case TPM_EO_UNSIGNED_GE:
            // compare A >= B unsigned
            return (UnsignedCompareB(size, opA, size, opB) >= 0);
            break;
        case TPM_EO_SIGNED_LE:
            // compare A <= B signed
            return (SignedCompareB(size, opA, size, opB) <= 0);
            break;
        case TPM_EO_UNSIGNED_LE:
            // compare A <= B unsigned
            return (UnsignedCompareB(size, opA, size, opB) <= 0);
            break;
        case TPM_EO_BITSET:
            // All bits SET in B are SET in A. ((A&B)=B)
            {
                UINT32 i;
                for(i = 0; i < size; i++)
                    if((opA[i] & opB[i]) != opB[i])
                        return FALSE;
            }
            break;
```

```
        case TPM_EO_BITCLEAR:
            // All bits SET in B are CLEAR in A. ((A&B)=0)
            {
                UINT32 i;
                for(i = 0; i < size; i++)
                    if((opA[i] & opB[i]) != 0)
                        return FALSE;
            }
            break;
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    return TRUE;
}
```

7.64 /tpm/src/command/Ecdaa/Commit.c

```
#include "Tpm.h"
#include "Commit_fp.h"
#include "TpmMath_Util_fp.h"

#if CC_Commit  // Conditional expansion of this file

/*(See part 3 specification)
// This command performs the point multiply operations for anonymous signing
// scheme.
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'keyHandle' references a restricted key that is not a
//                              signing key
//      TPM_RC_ECC_POINT        either 'P1' or the point derived from 's2' is not on
//                              the curve of 'keyHandle'
//      TPM_RC_HASH             invalid name algorithm in 'keyHandle'
//      TPM_RC_KEY              'keyHandle' does not reference an ECC key
//      TPM_RC_SCHEME           the scheme of 'keyHandle' is not an anonymous scheme
//      TPM_RC_NO_RESULT        'K', 'L' or 'E' was a point at infinity; or
//                              failed to generate "r" value
//      TPM_RC_SIZE             's2' is empty but 'y2' is not or 's2' provided but
//                              'y2' is not
TPM_RC
TPM2_Commit(Commit_In*  in,  // IN: input parameter list
            Commit_Out* out  // OUT: output parameter list
)
{
    OBJECT*              eccKey;
    TPMS_ECC_POINT       P2;
    TPMS_ECC_POINT*      pP2 = NULL;
    TPMS_ECC_POINT*      pP1 = NULL;
    TPM2B_ECC_PARAMETER r;
    TPM2B_ECC_PARAMETER p;
    TPM_RC               result;
    TPMS_ECC_PARMS*      parms;

    // Input Validation

    eccKey = HandleToObject(in->signHandle);
    parms  = &eccKey->publicArea.parameters.eccDetail;

    // Input key must be an ECC key
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RCS_KEY + RC_Commit_signHandle;

    // This command may only be used with a sign-only key using an anonymous
    // scheme.
```

```
    // NOTE: a sign + decrypt key has no scheme so it will not be an anonymous one
    // and an unrestricted sign key might no have a signing scheme but it can't
    // be use in Commit()
    if(!CryptIsSchemeAnonymous(parms->scheme.scheme))
        return TPM_RCS_SCHEME + RC_Commit_signHandle;

    // Make sure that both parts of P2 are present if either is present
    if((in->s2.t.size == 0) != (in->y2.t.size == 0))
        return TPM_RCS_SIZE + RC_Commit_y2;

    // Get prime modulus for the curve. This is needed later but getting this now
    // allows confirmation that the curve exists.
    if(!TpmMath_IntTo2B(ExtEcc_CurveGetPrime(parms->curveID), &p.b, 0))
        return TPM_RCS_KEY + RC_Commit_signHandle;

    // Get the random value that will be used in the point multiplications
    // Note: this does not commit the count.
    if(!CryptGenerateR(&r, NULL, parms->curveID, &eccKey->name))
        return TPM_RC_NO_RESULT;

    // Set up P2 if s2 and Y2 are provided
    if(in->s2.t.size != 0)
    {
        TPM2B_DIGEST x2;

        pP2 = &P2;

        // copy y2 for P2
        P2.y = in->y2;

        // Compute x2  HnameAlg(s2) mod p
        //      do the hash operation on s2 with the size of curve 'p'
        x2.t.size = CryptHashBlock(eccKey->publicArea.nameAlg,
                                   in->s2.t.size,
                                   in->s2.t.buffer,
                                   sizeof(x2.t.buffer),
                                   x2.t.buffer);

        // If there were error returns in the hash routine, indicate a problem
        // with the hash algorithm selection
        if(x2.t.size == 0)
            return TPM_RCS_HASH + RC_Commit_signHandle;
        // The size of the remainder will be same as the size of p. DivideB() will
        // pad the results (leading zeros) if necessary to make the size the same
        P2.x.t.size = p.t.size;
        //  set p2.x = hash(s2) mod p
        if(DivideB(&x2.b, &p.b, NULL, &P2.x.b) != TPM_RC_SUCCESS)
            return TPM_RC_NO_RESULT;

        if(!CryptEccIsPointOnCurve(parms->curveID, pP2))
            return TPM_RCS_ECC_POINT + RC_Commit_s2;

        if(eccKey->attributes.publicOnly == SET)
            return TPM_RCS_KEY + RC_Commit_signHandle;
    }
    // If there is a P1, make sure that it is on the curve
    // NOTE: an "empty" point has two UINT16 values which are the size values
    // for each of the coordinates.
    if(in->P1.size > 4)
    {
        pP1 = &in->P1.point;
        if(!CryptEccIsPointOnCurve(parms->curveID, pP1))
            return TPM_RCS_ECC_POINT + RC_Commit_P1;
    }

    // Pass the parameters to CryptCommit.
```

```
    // The work is not done in-line because it does several point multiplies
    // with the same curve.  It saves work by not having to reload the curve
    // parameters multiple times.
    result = CryptEccCommitCompute(&out->K.point,
                                   &out->L.point,
                                   &out->E.point,
                                   parms->curveID,
                                   pP1,
                                   pP2,
                                   &eccKey->sensitive.sensitive.ecc,
                                   &r);
    if(result != TPM_RC_SUCCESS)
        return result;

    // The commit computation was successful so complete the commit by setting
    // the bit
    out->counter = CryptCommit();

    return TPM_RC_SUCCESS;
}

#endif  // CC_Commit
```

7.65 /tpm/src/command/FieldUpgrade/FieldUpgradeData.c

```
#include "Tpm.h"
#include "FieldUpgradeData_fp.h"
#if CC_FieldUpgradeData  // Conditional expansion of this file

/*(See part 3 specification)
// FieldUpgradeData
*/
TPM_RC
TPM2_FieldUpgradeData(FieldUpgradeData_In*  in,  // IN: input parameter list
                      FieldUpgradeData_Out* out  // OUT: output parameter list
)
{
    // Not implemented
    UNUSED_PARAMETER(in);
    UNUSED_PARAMETER(out);
    return TPM_RC_SUCCESS;
}
#endif
```

7.66 /tpm/src/command/FieldUpgrade/FieldUpgradeStart.c

```
#include "Tpm.h"
#include "FieldUpgradeStart_fp.h"
#if CC_FieldUpgradeStart  // Conditional expansion of this file

/*(See part 3 specification)
// FieldUpgradeStart
*/
TPM_RC
TPM2_FieldUpgradeStart(FieldUpgradeStart_In* in  // IN: input parameter list
)
{
    // Not implemented
    UNUSED_PARAMETER(in);
    return TPM_RC_SUCCESS;
}
#endif
```

7.67 /tpm/src/command/FieldUpgrade/FirmwareRead.c

```
#include "Tpm.h"
#include "FirmwareRead_fp.h"

#if CC_FirmwareRead  // Conditional expansion of this file

/*(See part 3 specification)
// FirmwareRead
*/
TPM_RC
TPM2_FirmwareRead(FirmwareRead_In*  in,  // IN: input parameter list
                  FirmwareRead_Out* out  // OUT: output parameter list
)
{
    // Not implemented
    UNUSED_PARAMETER(in);
    UNUSED_PARAMETER(out);
    return TPM_RC_SUCCESS;
}

#endif  // CC_FirmwareRead
```

7.68 /tpm/src/command/HashHMAC/EventSequenceComplete.c

```
#include "Tpm.h"
#include "EventSequenceComplete_fp.h"

#if CC_EventSequenceComplete  // Conditional expansion of this file

/*(See part 3 specification)
  Complete an event sequence and flush the object.
*/
//  Return Type: TPM_RC
//      TPM_RC_LOCALITY     PCR extension is not allowed at the current locality
//      TPM_RC_MODE         input handle is not a valid event sequence object
TPM_RC
TPM2_EventSequenceComplete(
    EventSequenceComplete_In*  in,  // IN: input parameter list
    EventSequenceComplete_Out* out  // OUT: output parameter list
)
{
    HASH_OBJECT* hashObject;
    UINT32       i;
    TPM_ALG_ID   hashAlg;
    // Input validation
    // get the event sequence object pointer
    hashObject = (HASH_OBJECT*)HandleToObject(in->sequenceHandle);

    // input handle must reference an event sequence object
    if(hashObject->attributes.eventSeq != SET)
        return TPM_RCS_MODE + RC_EventSequenceComplete_sequenceHandle;

    // see if a PCR extend is requested in call
    if(in->pcrHandle != TPM_RH_NULL)
    {
        // see if extend of the PCR is allowed at the locality of the command,
        if(!PCRIsExtendAllowed(in->pcrHandle))
            return TPM_RC_LOCALITY;
        // if an extend is going to take place, then check to see if there has
        // been an orderly shutdown. If so, and the selected PCR is one of the
        // state saved PCR, then the orderly state has to change. The orderly state
        // does not change for PCR that are not preserved.
        // NOTE: This doesn't just check for Shutdown(STATE) because the orderly
        // state will have to change if this is a state-saved PCR regardless
```

```
        // of the current state. This is because a subsequent Shutdown(STATE) will
        // check to see if there was an orderly shutdown and not do anything if
        // there was. So, this must indicate that a future Shutdown(STATE) has
        // something to do.
        if(PCRIsStateSaved(in->pcrHandle))
            RETURN_IF_ORDERLY;
    }
    // Command Output
    out->results.count = 0;

    for(i = 0; i < HASH_COUNT; i++)
    {
        hashAlg = CryptHashGetAlgByIndex(i);
        // Update last piece of data
        CryptDigestUpdate2B(&hashObject->state.hashState[i], &in->buffer.b);
        // Complete hash
        out->results.digests[out->results.count].hashAlg = hashAlg;
        CryptHashEnd(&hashObject->state.hashState[i],
                     CryptHashGetDigestSize(hashAlg),
                     (BYTE*)&out->results.digests[out->results.count].digest);
        // Extend PCR
        if(in->pcrHandle != TPM_RH_NULL)
            PCRExtend(in->pcrHandle,
                      hashAlg,
                      CryptHashGetDigestSize(hashAlg),
                      (BYTE*)&out->results.digests[out->results.count].digest);
        out->results.count++;
    }
    // Internal Data Update
    // mark sequence object as evict so it will be flushed on the way out
    hashObject->attributes.evict = SET;

    return TPM_RC_SUCCESS;
}

#endif  // CC_EventSequenceComplete
```

7.69 /tpm/src/command/HashHMAC/HashSequenceStart.c

```
#include "Tpm.h"
#include "HashSequenceStart_fp.h"

#if CC_HashSequenceStart  // Conditional expansion of this file

/*(See part 3 specification)
// Start a hash or an event sequence
*/
//  Return Type: TPM_RC
//      TPM_RC_OBJECT_MEMORY        no space to create an internal object
TPM_RC
TPM2_HashSequenceStart(HashSequenceStart_In*  in,  // IN: input parameter list
                       HashSequenceStart_Out* out  // OUT: output parameter list
)
{
    // Internal Data Update

    if(in->hashAlg == TPM_ALG_NULL)
        // Start a event sequence.  A TPM_RC_OBJECT_MEMORY error may be
        // returned at this point
        return ObjectCreateEventSequence(&in->auth, &out->sequenceHandle);

    // Start a hash sequence.  A TPM_RC_OBJECT_MEMORY error may be
    // returned at this point
    return ObjectCreateHashSequence(in->hashAlg, &in->auth, &out->sequenceHandle);
}
```

```
#endif  // CC_HashSequenceStart
```

7.70 /tpm/src/command/HashHMAC/HMAC_Start.c

```
#include "Tpm.h"
#include "HMAC_Start_fp.h"

#if CC_HMAC_Start  // Conditional expansion of this file

/*(See part 3 specification)
// Initialize a HMAC sequence and create a sequence object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       key referenced by 'handle' is not a signing key
//                              or is restricted
//      TPM_RC_OBJECT_MEMORY    no space to create an internal object
//      TPM_RC_KEY              key referenced by 'handle' is not an HMAC key
//      TPM_RC_VALUE            'hashAlg' is not compatible with the hash algorithm
//                              of the scheme of the object referenced by 'handle'
TPM_RC
TPM2_HMAC_Start(HMAC_Start_In*  in,  // IN: input parameter list
                HMAC_Start_Out* out  // OUT: output parameter list
)
{
    OBJECT*      keyObject;
    TPMT_PUBLIC* publicArea;
    TPM_ALG_ID   hashAlg;

    // Input Validation

    // Get HMAC key object and public area pointers
    keyObject  = HandleToObject(in->handle);
    publicArea = &keyObject->publicArea;

    // Make sure that the key is an HMAC key
    if(publicArea->type != TPM_ALG_KEYEDHASH)
        return TPM_RCS_TYPE + RC_HMAC_Start_handle;

    // and that it is unrestricted
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted))
        return TPM_RCS_ATTRIBUTES + RC_HMAC_Start_handle;

    // and that it is a signing key
    if(!IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
        return TPM_RCS_KEY + RC_HMAC_Start_handle;

    // See if the key has a default
    if(publicArea->parameters.keyedHashDetail.scheme.scheme == TPM_ALG_NULL)
        // it doesn't so use the input value
        hashAlg = in->hashAlg;
    else
    {
        // key has a default so use it
        hashAlg = publicArea->parameters.keyedHashDetail.scheme.details.hmac.hashAlg;
        // and verify that the input was either the  TPM_ALG_NULL or the default
        if(in->hashAlg != TPM_ALG_NULL && in->hashAlg != hashAlg)
            hashAlg = TPM_ALG_NULL;
    }
    // if we ended up without a hash algorithm then return an error
    if(hashAlg == TPM_ALG_NULL)
        return TPM_RCS_VALUE + RC_HMAC_Start_hashAlg;

    // Internal Data Update
```

```
    // Create a HMAC sequence object. A TPM_RC_OBJECT_MEMORY error may be
    // returned at this point
    return ObjectCreateHMACSequence(
        hashAlg, keyObject, &in->auth, &out->sequenceHandle);
}

#endif  // CC_HMAC_Start
```

7.71 /tpm/src/command/HashHMAC/MAC_Start.c

```
#include "Tpm.h"
#include "MAC_Start_fp.h"

#if CC_MAC_Start  // Conditional expansion of this file

/*(See part 3 specification)
// Initialize a HMAC sequence and create a sequence object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       key referenced by 'handle' is not a signing key
//                              or is restricted
//      TPM_RC_OBJECT_MEMORY    no space to create an internal object
//      TPM_RC_KEY              key referenced by 'handle' is not an HMAC key
//      TPM_RC_VALUE            'hashAlg' is not compatible with the hash algorithm
//                              of the scheme of the object referenced by 'handle'
TPM_RC
TPM2_MAC_Start(MAC_Start_In*  in,  // IN: input parameter list
               MAC_Start_Out* out  // OUT: output parameter list
)
{
    OBJECT*      keyObject;
    TPMT_PUBLIC* publicArea;
    TPM_RC       result;

    // Input Validation

    // Get HMAC key object and public area pointers
    keyObject  = HandleToObject(in->handle);
    publicArea = &keyObject->publicArea;

    // Make sure that the key can do what is required
    result = CryptSelectMac(publicArea, &in->inScheme);
    // If the key is not able to do a MAC, indicate that the handle selects an
    // object that can't do a MAC
    if(result == TPM_RCS_TYPE)
        return TPM_RCS_TYPE + RC_MAC_Start_handle;
    // If there is another error type, indicate that the scheme and key are not
    // compatible
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_MAC_Start_inScheme);
    // Make sure that the key is not restricted
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted))
        return TPM_RCS_ATTRIBUTES + RC_MAC_Start_handle;
    // and that it is a signing key
    if(!IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
        return TPM_RCS_KEY + RC_MAC_Start_handle;

    // Internal Data Update
    // Create a HMAC sequence object. A TPM_RC_OBJECT_MEMORY error may be
    // returned at this point
    return ObjectCreateHMACSequence(
        in->inScheme, keyObject, &in->auth, &out->sequenceHandle);
}

#endif  // CC_MAC_Start
```

7.72 /tpm/src/command/HashHMAC/SequenceComplete.c

```
#include "Tpm.h"
#include "SequenceComplete_fp.h"

#if CC_SequenceComplete  // Conditional expansion of this file

/*(See part 3 specification)
// Complete a sequence and flush the object.
*/
//  Return Type: TPM_RC
//      TPM_RC_MODE             'sequenceHandle' does not reference a hash or HMAC
//                              sequence object
TPM_RC
TPM2_SequenceComplete(SequenceComplete_In*  in,  // IN: input parameter list
                      SequenceComplete_Out* out  // OUT: output parameter list
)
{
    HASH_OBJECT* hashObject;
    // Input validation
    // Get hash object pointer
    hashObject = (HASH_OBJECT*)HandleToObject(in->sequenceHandle);

    // input handle must be a hash or HMAC sequence object.
    if(hashObject->attributes.hashSeq == CLEAR
       && hashObject->attributes.hmacSeq == CLEAR)
        return TPM_RCS_MODE + RC_SequenceComplete_sequenceHandle;
    // Command Output
    if(hashObject->attributes.hashSeq == SET)  // sequence object for hash
    {
        // Get the hash algorithm before the algorithm is lost in CryptHashEnd
        TPM_ALG_ID hashAlg = hashObject->state.hashState[0].hashAlg;

        // Update last piece of the data
        CryptDigestUpdate2B(&hashObject->state.hashState[0], &in->buffer.b);

        // Complete hash
        out->result.t.size = CryptHashEnd(&hashObject->state.hashState[0],
                                          sizeof(out->result.t.buffer),
                                          out->result.t.buffer);
        // Check if the first block of the sequence has been received
        if(hashObject->attributes.firstBlock == CLEAR)
        {
            // If not, then this is the first block so see if it is 'safe'
            // to sign.
            if(TicketIsSafe(&in->buffer.b))
                hashObject->attributes.ticketSafe = SET;
        }
        // Output ticket
        out->validation.tag       = TPM_ST_HASHCHECK;
        out->validation.hierarchy = in->hierarchy;

        if(in->hierarchy == TPM_RH_NULL)
        {
            // Ticket is not required
            out->validation.digest.t.size = 0;
        }
        else if(hashObject->attributes.ticketSafe == CLEAR)
        {
            // Ticket is not safe to generate
            out->validation.hierarchy     = TPM_RH_NULL;
            out->validation.digest.t.size = 0;
        }
        else
        {
            TPM_RC result;
```

```
            // Compute ticket
            result = TicketComputeHashCheck(
                out->validation.hierarchy, hashAlg, &out->result, &out->validation);
            if(result != TPM_RC_SUCCESS)
                return result;
        }
    }
    else
    {
        //   Update last piece of data
        CryptDigestUpdate2B(&hashObject->state.hmacState.hashState, &in->buffer.b);
#   if !SMAC_IMPLEMENTED
        // Complete HMAC
        out->result.t.size = CryptHmacEnd(&(hashObject->state.hmacState),
                                          sizeof(out->result.t.buffer),
                                          out->result.t.buffer);
#   else
        // Complete the MAC
        out->result.t.size = CryptMacEnd(&hashObject->state.hmacState,
                                         sizeof(out->result.t.buffer),
                                         out->result.t.buffer);
#   endif
        // No ticket is generated for HMAC sequence
        out->validation.tag           = TPM_ST_HASHCHECK;
        out->validation.hierarchy     = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    // Internal Data Update
    // mark sequence object as evict so it will be flushed on the way out
    hashObject->attributes.evict = SET;

    return TPM_RC_SUCCESS;
}

#endif  // CC_SequenceComplete
```

7.73 /tpm/src/command/HashHMAC/SequenceUpdate.c

```
#include "Tpm.h"
#include "SequenceUpdate_fp.h"

#if CC_SequenceUpdate  // Conditional expansion of this file

/*(See part 3 specification)
// This function is used to add data to a sequence object.
*/
//  Return Type: TPM_RC
//      TPM_RC_MODE             'sequenceHandle' does not reference a hash or HMAC
//                              sequence object
TPM_RC
TPM2_SequenceUpdate(SequenceUpdate_In* in  // IN: input parameter list
)
{
    OBJECT*      object;
    HASH_OBJECT* hashObject;

    // Input Validation

    // Get sequence object pointer
    object     = HandleToObject(in->sequenceHandle);
    hashObject = (HASH_OBJECT*)object;

    // Check that referenced object is a sequence object.
    if(!ObjectIsSequence(object))
        return TPM_RCS_MODE + RC_SequenceUpdate_sequenceHandle;
```

```
    // Internal Data Update

    if(object->attributes.eventSeq == SET)
    {
        // Update event sequence object
        UINT32 i;
        for(i = 0; i < HASH_COUNT; i++)
        {
            // Update sequence object
            CryptDigestUpdate2B(&hashObject->state.hashState[i], &in->buffer.b);
        }
    }
    else
    {
        // Update hash/HMAC sequence object
        if(hashObject->attributes.hashSeq == SET)
        {
            // Is this the first block of the sequence
            if(hashObject->attributes.firstBlock == CLEAR)
            {
                // If so, indicate that first block was received
                hashObject->attributes.firstBlock = SET;

                // Check the first block to see if the first block can contain
                // the TPM_GENERATED_VALUE.  If it does, it is not safe for
                // a ticket.
                if(TicketIsSafe(&in->buffer.b))
                    hashObject->attributes.ticketSafe = SET;
            }
            // Update sequence object hash/HMAC stack
            CryptDigestUpdate2B(&hashObject->state.hashState[0], &in->buffer.b);
        }
        else if(object->attributes.hmacSeq == SET)
        {
            // Update sequence object HMAC stack
            CryptDigestUpdate2B(&hashObject->state.hmacState.hashState,
                                &in->buffer.b);
        }
    }
    return TPM_RC_SUCCESS;
}

#endif  // CC_SequenceUpdate
```

7.74 /tpm/src/command/Hierarchy/ChangeEPS.c

```
#include "Tpm.h"
#include "ChangeEPS_fp.h"

#if CC_ChangeEPS  // Conditional expansion of this file

/*(See part 3 specification)
// Reset current EPS value
*/
TPM_RC
TPM2_ChangeEPS(ChangeEPS_In* in  // IN: input parameter list
)
{
    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Input parameter is not reference in command action
```

```
    NOT_REFERENCED(in);

    // Internal Data Update

    // Reset endorsement hierarchy seed from RNG
    CryptRandomGenerate(sizeof(gp.EPSeed.t.buffer), gp.EPSeed.t.buffer);

    // Create new ehProof value from RNG
    CryptRandomGenerate(sizeof(gp.ehProof.t.buffer), gp.ehProof.t.buffer);

    // Enable endorsement hierarchy
    gc.ehEnable = TRUE;

    // set authValue buffer to zeros
    MemorySet(gp.endorsementAuth.t.buffer, 0, gp.endorsementAuth.t.size);
    // Set endorsement authValue to null
    gp.endorsementAuth.t.size = 0;

    // Set endorsement authPolicy to null
    gp.endorsementAlg           = TPM_ALG_NULL;
    gp.endorsementPolicy.t.size = 0;

    // Flush loaded object in endorsement hierarchy
    ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);

    // Flush evict object of endorsement hierarchy stored in NV
    NvFlushHierarchy(TPM_RH_ENDORSEMENT);

    // Save hierarchy changes to NV
    NV_SYNC_PERSISTENT(EPSeed);
    NV_SYNC_PERSISTENT(ehProof);
    NV_SYNC_PERSISTENT(endorsementAuth);
    NV_SYNC_PERSISTENT(endorsementAlg);
    NV_SYNC_PERSISTENT(endorsementPolicy);

    // orderly state should be cleared because of the update to state clear data
    g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
}

#endif  // CC_ChangeEPS
```

7.75 /tpm/src/command/Hierarchy/ChangePPS.c

```
#include "Tpm.h"
#include "ChangePPS_fp.h"

#if CC_ChangePPS  // Conditional expansion of this file

/*(See part 3 specification)
// Reset current PPS value
*/
TPM_RC
TPM2_ChangePPS(ChangePPS_In* in  // IN: input parameter list
)
{
    UINT32 i;

    // Check if NV is available.  A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
    // error may be returned at this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Input parameter is not reference in command action
    NOT_REFERENCED(in);
```

```
    // Internal Data Update

    // Reset platform hierarchy seed from RNG
    CryptRandomGenerate(sizeof(gp.PPSeed.t.buffer), gp.PPSeed.t.buffer);

    // Create a new phProof value from RNG to prevent the saved platform
    // hierarchy contexts being loaded
    CryptRandomGenerate(sizeof(gp.phProof.t.buffer), gp.phProof.t.buffer);

    // Set platform authPolicy to null
    gc.platformAlg           = TPM_ALG_NULL;
    gc.platformPolicy.t.size = 0;

    // Flush loaded object in platform hierarchy
    ObjectFlushHierarchy(TPM_RH_PLATFORM);

    // Flush platform evict object and index in NV
    NvFlushHierarchy(TPM_RH_PLATFORM);

    // Save hierarchy changes to NV
    NV_SYNC_PERSISTENT(PPSeed);
    NV_SYNC_PERSISTENT(phProof);

    // Re-initialize PCR policies
#   if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
    for(i = 0; i < NUM_POLICY_PCR_GROUP; i++)
    {
        gp.pcrPolicies.hashAlg[i]       = TPM_ALG_NULL;
        gp.pcrPolicies.policy[i].t.size = 0;
    }
    NV_SYNC_PERSISTENT(pcrPolicies);
#   endif

    // orderly state should be cleared because of the update to state clear data
    g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
}

#endif  // CC_ChangePPS
```

7.76 /tpm/src/command/Hierarchy/Clear.c

```
#include "Tpm.h"
#include "Clear_fp.h"

#if CC_Clear  // Conditional expansion of this file

/*(See part 3 specification)
// Clear owner
*/
//  Return Type: TPM_RC
//      TPM_RC_DISABLED             Clear command has been disabled
TPM_RC
TPM2_Clear(Clear_In* in  // IN: input parameter list
)
{
    // Input parameter is not reference in command action
    NOT_REFERENCED(in);

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;
```

```
    // Input Validation

    // If Clear command is disabled, return an error
    if(gp.disableClear)
        return TPM_RC_DISABLED;

    // Internal Data Update

    // Reset storage hierarchy seed from RNG
    CryptRandomGenerate(sizeof(gp.SPSeed.t.buffer), gp.SPSeed.t.buffer);

    // Create new shProof and ehProof value from RNG
    CryptRandomGenerate(sizeof(gp.shProof.t.buffer), gp.shProof.t.buffer);
    CryptRandomGenerate(sizeof(gp.ehProof.t.buffer), gp.ehProof.t.buffer);

    // Enable storage and endorsement hierarchy
    gc.shEnable = gc.ehEnable = TRUE;

    // set the authValue buffers to zero
    MemorySet(&gp.ownerAuth, 0, sizeof(gp.ownerAuth));
    MemorySet(&gp.endorsementAuth, 0, sizeof(gp.endorsementAuth));
    MemorySet(&gp.lockoutAuth, 0, sizeof(gp.lockoutAuth));

    // Set storage, endorsement, and lockout authPolicy to null
    gp.ownerAlg = gp.endorsementAlg = gp.lockoutAlg = TPM_ALG_NULL;
    MemorySet(&gp.ownerPolicy, 0, sizeof(gp.ownerPolicy));
    MemorySet(&gp.endorsementPolicy, 0, sizeof(gp.endorsementPolicy));
    MemorySet(&gp.lockoutPolicy, 0, sizeof(gp.lockoutPolicy));

    // Flush loaded object in storage and endorsement hierarchy
    ObjectFlushHierarchy(TPM_RH_OWNER);
    ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);

    // Flush owner and endorsement object and owner index in NV
    NvFlushHierarchy(TPM_RH_OWNER);
    NvFlushHierarchy(TPM_RH_ENDORSEMENT);

    // Initialize dictionary attack parameters
    DAPreInstall_Init();

    // Reset clock
    go.clock     = 0;
    go.clockSafe = YES;
    NvWrite(NV_ORDERLY_DATA, sizeof(ORDERLY_DATA), &go);

    // Reset counters
    gp.resetCount = gr.restartCount = gr.clearCount = 0;
    gp.auditCounter                                 = 0;

    // Save persistent data changes to NV
    // Note: since there are so many changes to the persistent data structure, the
    // entire PERSISTENT_DATA structure is written as a unit
    NvWrite(NV_PERSISTENT_DATA, sizeof(PERSISTENT_DATA), &gp);

    // Reset the PCR authValues (this does not change the PCRs)
    PCR_ClearAuth();

    // Bump the PCR counter
    PCRChanged(0);

    // orderly state should be cleared because of the update to state clear data
    g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
}
```

```
#endif  // CC_Clear
```

7.77 /tpm/src/command/Hierarchy/ClearControl.c

```
#include "Tpm.h"
#include "ClearControl_fp.h"

#if CC_ClearControl  // Conditional expansion of this file

/*(See part 3 specification)
// Enable or disable the execution of TPM2_Clear command
*/
//  Return Type: TPM_RC
//      TPM_RC_AUTH_FAIL            authorization is not properly given
TPM_RC
TPM2_ClearControl(ClearControl_In* in  // IN: input parameter list
)
{
    // The command needs NV update.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Input Validation

    // LockoutAuth may be used to set disableLockoutClear to TRUE but not to FALSE
    if(in->auth == TPM_RH_LOCKOUT && in->disable == NO)
        return TPM_RC_AUTH_FAIL;

    // Internal Data Update

    if(in->disable == YES)
        gp.disableClear = TRUE;
    else
        gp.disableClear = FALSE;

    // Record the change to NV
    NV_SYNC_PERSISTENT(disableClear);

    return TPM_RC_SUCCESS;
}

#endif  // CC_ClearControl
```

7.78 /tpm/src/command/Hierarchy/CreatePrimary.c

```
#include "Tpm.h"
#include "CreatePrimary_fp.h"

#if CC_CreatePrimary  // Conditional expansion of this file

/*(See part 3 specification)
// Creates a primary or temporary object from a primary seed.
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       sensitiveDataOrigin is CLEAR when sensitive.data is an
//                              Empty Buffer; 'fixedTPM', 'fixedParent', or
//                              'encryptedDuplication' attributes are inconsistent
//                              between themselves or with those of the parent object;
//                              inconsistent 'restricted', 'decrypt', 'sign',
//                              'firmwareLimited', or 'svnLimited' attributes;
//                              attempt to inject sensitive data for an asymmetric
//                              key;
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
```

```
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM failed
//                              to derive the Firmware SVN Secret for the requested
//                              SVN.
//      TPM_RC_KDF              incorrect KDF specified for decrypting keyed hash
//                              object
//      TPM_RC_KEY              a provided symmetric key value is not allowed
//      TPM_RC_OBJECT_MEMORY    there is no free slot for the object
//      TPM_RC_SCHEME           inconsistent attributes 'decrypt', 'sign',
//                              'restricted' and key's scheme ID; or hash algorithm is
//                              inconsistent with the scheme ID for keyed hash object
//      TPM_RC_SIZE             size of public authorization policy or sensitive
//                              authorization value does not match digest size of the
//                              name algorithm; or sensitive data size for the keyed
//                              hash object is larger than is allowed for the scheme
//      TPM_RC_SYMMETRIC        a storage key with no symmetric algorithm specified;
//                              or non-storage key with symmetric algorithm different
//                              from TPM_ALG_NULL
//      TPM_RC_TYPE             unknown object type
TPM_RC
TPM2_CreatePrimary(CreatePrimary_In*  in,  // IN: input parameter list
                   CreatePrimary_Out* out  // OUT: output parameter list
)
{
    TPM_RC       result = TPM_RC_SUCCESS;
    TPMT_PUBLIC* publicArea;
    DRBG_STATE   rand;
    OBJECT*      newObject;
    TPM2B_NAME   name;
    TPM2B_SEED   primary_seed;

    // Input Validation
    // Will need a place to put the result
    newObject = FindEmptyObjectSlot(&out->objectHandle);
    if(newObject == NULL)
        return TPM_RC_OBJECT_MEMORY;
    // Get the address of the public area in the new object
    // (this is just to save typing)
    publicArea  = &newObject->publicArea;

    *publicArea = in->inPublic.publicArea;

    // Check attributes in input public area. CreateChecks() checks the things that
    // are unique to creation and then validates the attributes and values that are
    // common to create and load.
    result = CreateChecks(
        NULL, in->primaryHandle, publicArea, in->inSensitive.sensitive.data.t.size);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_CreatePrimary_inPublic);
    // Validate the sensitive area values
    if(!AdjustAuthSize(&in->inSensitive.sensitive.userAuth, publicArea->nameAlg))
        return TPM_RCS_SIZE + RC_CreatePrimary_inSensitive;
    // Command output
    // Compute the name using out->name as a scratch area (this is not the value
    // that ultimately will be returned, then instantiate the state that will be
    // used as a random number generator during the object creation.
    // The caller does not know the seed values so the actual name does not have
    // to be over the input, it can be over the unmarshaled structure.

    result = HierarchyGetPrimarySeed(in->primaryHandle, &primary_seed);
    if(result != TPM_RC_SUCCESS)
        return result;

    result =
        DRBG_InstantiateSeeded(&rand,
```

```
                                &primary_seed.b,
                                PRIMARY_OBJECT_CREATION,
                                (TPM2B*)PublicMarshalAndComputeName(publicArea, &name),
                                &in->inSensitive.sensitive.data.b);
    MemorySet(primary_seed.b.buffer, 0, primary_seed.b.size);

    if(result == TPM_RC_SUCCESS)
    {
        newObject->attributes.primary = SET;
        if(HierarchyNormalizeHandle(in->primaryHandle) == TPM_RH_ENDORSEMENT)
            newObject->attributes.epsHierarchy = SET;

        // Create the primary object.
        result = CryptCreateObject(
            newObject, &in->inSensitive.sensitive, (RAND_STATE*)&rand);
        DRBG_Uninstantiate(&rand);
    }
    if(result != TPM_RC_SUCCESS)
        return result;

    // Set the publicArea and name from the computed values
    out->outPublic.publicArea = newObject->publicArea;
    out->name                 = newObject->name;

    // Fill in creation data
    FillInCreationData(in->primaryHandle,
                       publicArea->nameAlg,
                       &in->creationPCR,
                       &in->outsideInfo,
                       &out->creationData,
                       &out->creationHash);

    // Compute creation ticket
    result = TicketComputeCreation(EntityGetHierarchy(in->primaryHandle),
                                   &out->name,
                                   &out->creationHash,
                                   &out->creationTicket);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Set the remaining attributes for a loaded object
    ObjectSetLoadedAttributes(newObject, in->primaryHandle);
    return result;
}

#endif  // CC_CreatePrimary
```

7.79 /tpm/src/command/Hierarchy/HierarchyChangeAuth.c

```
#include "Tpm.h"
#include "HierarchyChangeAuth_fp.h"

#if CC_HierarchyChangeAuth  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// Set a hierarchy authValue
*/
//  Return Type: TPM_RC
//      TPM_RC_SIZE         'newAuth' size is greater than that of integrity hash
//                           digest
TPM_RC
TPM2_HierarchyChangeAuth(HierarchyChangeAuth_In* in  // IN: input parameter list
)
```

```
{
    // The command needs NV update.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Make sure that the authorization value is a reasonable size (not larger than
    // the size of the digest produced by the integrity hash. The integrity
    // hash is assumed to produce the longest digest of any hash implemented
    // on the TPM. This will also remove trailing zeros from the authValue.
    if(MemoryRemoveTrailingZeros(&in->newAuth) > CONTEXT_INTEGRITY_HASH_SIZE)
        return TPM_RCS_SIZE + RC_HierarchyChangeAuth_newAuth;

    // Set hierarchy authValue
    switch(in->authHandle)
    {
        case TPM_RH_OWNER:
            gp.ownerAuth = in->newAuth;
            NV_SYNC_PERSISTENT(ownerAuth);
            break;
        case TPM_RH_ENDORSEMENT:
            gp.endorsementAuth = in->newAuth;
            NV_SYNC_PERSISTENT(endorsementAuth);
            break;
        case TPM_RH_PLATFORM:
            gc.platformAuth = in->newAuth;
            // orderly state should be cleared
            g_clearOrderly = TRUE;
            break;
        case TPM_RH_LOCKOUT:
            gp.lockoutAuth = in->newAuth;
            NV_SYNC_PERSISTENT(lockoutAuth);
            break;
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_HierarchyChangeAuth
```

7.80 /tpm/src/command/Hierarchy/HierarchyControl.c

```
#include "Tpm.h"
#include "HierarchyControl_fp.h"

#if CC_HierarchyControl  // Conditional expansion of this file

/*(See part 3 specification)
// Enable or disable use of a hierarchy
*/
//  Return Type: TPM_RC
//      TPM_RC_AUTH_TYPE        'authHandle' is not applicable to 'hierarchy' in its
//                              current state
TPM_RC
TPM2_HierarchyControl(HierarchyControl_In* in  // IN: input parameter list
)
{
    BOOL  select   = (in->state == YES);
    BOOL* selected = NULL;

    // Input Validation
    switch(in->enable)
    {
        // Platform hierarchy has to be disabled by PlatformAuth
```

```
    // If the platform hierarchy has already been disabled, only a reboot
    // can enable it again
    case TPM_RH_PLATFORM:
    case TPM_RH_PLATFORM_NV:
        if(in->authHandle != TPM_RH_PLATFORM)
            return TPM_RC_AUTH_TYPE;
        break;

    // ShEnable may be disabled if PlatformAuth/PlatformPolicy or
    // OwnerAuth/OwnerPolicy is provided.  If ShEnable is disabled, then it
    // may only be enabled if PlatformAuth/PlatformPolicy is provided.
    case TPM_RH_OWNER:
        if(in->authHandle != TPM_RH_PLATFORM && in->authHandle != TPM_RH_OWNER)
            return TPM_RC_AUTH_TYPE;
        if(gc.shEnable == FALSE && in->state == YES
           && in->authHandle != TPM_RH_PLATFORM)
            return TPM_RC_AUTH_TYPE;
        break;

    // EhEnable may be disabled if either PlatformAuth/PlatformPolicy or
    // EndosementAuth/EndorsementPolicy is provided.  If EhEnable is disabled,
    // then it may only be enabled if PlatformAuth/PlatformPolicy is
    // provided.
    case TPM_RH_ENDORSEMENT:
        if(in->authHandle != TPM_RH_PLATFORM
           && in->authHandle != TPM_RH_ENDORSEMENT)
            return TPM_RC_AUTH_TYPE;
        if(gc.ehEnable == FALSE && in->state == YES
           && in->authHandle != TPM_RH_PLATFORM)
            return TPM_RC_AUTH_TYPE;
        break;
    default:
        FAIL(FATAL_ERROR_INTERNAL);
        break;
}

// Internal Data Update

// Enable or disable the selected hierarchy
// Note: the authorization processing for this command may keep these
// command actions from being executed. For example, if phEnable is
// CLEAR, then platformAuth cannot be used for authorization. This
// means that would not be possible to use platformAuth to change the
// state of phEnable from CLEAR to SET.
// If it is decided that platformPolicy can still be used when phEnable
// is CLEAR, then this code could SET phEnable when proper platform
// policy is provided.
switch(in->enable)
{
    case TPM_RH_OWNER:
        selected = &gc.shEnable;
        break;
    case TPM_RH_ENDORSEMENT:
        selected = &gc.ehEnable;
        break;
    case TPM_RH_PLATFORM:
        selected = &g_phEnable;
        break;
    case TPM_RH_PLATFORM_NV:
        selected = &gc.phEnableNV;
        break;
    default:
        FAIL(FATAL_ERROR_INTERNAL);
        break;
}
if(selected != NULL && *selected != select)
```

```
    {
        // Before changing the internal state, make sure that NV is available.
        // Only need to update NV if changing the orderly state
        RETURN_IF_ORDERLY;

        // state is changing and NV is available so modify
        *selected = select;
        // If a hierarchy was just disabled, flush it
        if(select == CLEAR && in->enable != TPM_RH_PLATFORM_NV)
            // Flush hierarchy
            ObjectFlushHierarchy(in->enable);

        // orderly state should be cleared because of the update to state clear data
        // This gets processed in ExecuteCommand() on the way out.
        g_clearOrderly = TRUE;
    }
    return TPM_RC_SUCCESS;
}

#endif  // CC_HierarchyControl
```

7.81 /tpm/src/command/Hierarchy/SetPrimaryPolicy.c

```
#include "Tpm.h"
#include "SetPrimaryPolicy_fp.h"

#if CC_SetPrimaryPolicy  // Conditional expansion of this file

/*(See part 3 specification)
// Set a hierarchy policy
*/
//  Return Type: TPM_RC
//      TPM_RC_SIZE             size of input authPolicy is not consistent with
//                              input hash algorithm
TPM_RC
TPM2_SetPrimaryPolicy(SetPrimaryPolicy_In* in  // IN: input parameter list
)
{
    // Input Validation

    // Check the authPolicy consistent with hash algorithm. If the policy size is
    // zero, then the algorithm is required to be TPM_ALG_NULL
    if(in->authPolicy.t.size != CryptHashGetDigestSize(in->hashAlg))
        return TPM_RCS_SIZE + RC_SetPrimaryPolicy_authPolicy;

    // The command need NV update for OWNER and ENDORSEMENT hierarchy, and
    // might need orderlyState update for PLATFROM hierarchy.
    // Check if NV is available.  A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
    // error may be returned at this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Internal Data Update

    // Set hierarchy policy
    switch(in->authHandle)
    {
        case TPM_RH_OWNER:
            gp.ownerAlg    = in->hashAlg;
            gp.ownerPolicy = in->authPolicy;
            NV_SYNC_PERSISTENT(ownerAlg);
            NV_SYNC_PERSISTENT(ownerPolicy);
            break;
        case TPM_RH_ENDORSEMENT:
            gp.endorsementAlg    = in->hashAlg;
            gp.endorsementPolicy = in->authPolicy;
```

```
            NV_SYNC_PERSISTENT(endorsementAlg);
            NV_SYNC_PERSISTENT(endorsementPolicy);
            break;
        case TPM_RH_PLATFORM:
            gc.platformAlg    = in->hashAlg;
            gc.platformPolicy = in->authPolicy;
            // need to update orderly state
            g_clearOrderly = TRUE;
            break;
        case TPM_RH_LOCKOUT:
            gp.lockoutAlg    = in->hashAlg;
            gp.lockoutPolicy = in->authPolicy;
            NV_SYNC_PERSISTENT(lockoutAlg);
            NV_SYNC_PERSISTENT(lockoutPolicy);
            break;

#  if ACT_SUPPORT
#    define SET_ACT_POLICY(N)                          \
        case TPM_RH_ACT_##N:                           \
            go.ACT_##N.hashAlg    = in->hashAlg;    \
            go.ACT_##N.authPolicy = in->authPolicy; \
            g_clearOrderly        = TRUE;           \
            break;

            FOR_EACH_ACT(SET_ACT_POLICY)
#  endif  // ACT_SUPPORT

        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_SetPrimaryPolicy
```

7.82 /tpm/src/command/Misc/PP_Commands.c

```
#include "Tpm.h"
#include "PP_Commands_fp.h"

#if CC_PP_Commands  // Conditional expansion of this file

/*(See part 3 specification)
// This command is used to determine which commands require assertion of
// Physical Presence in addition to platformAuth/platformPolicy.
*/
TPM_RC
TPM2_PP_Commands(PP_Commands_In* in  // IN: input parameter list
)
{
    UINT32 i;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Internal Data Update

    // Process set list
    for(i = 0; i < in->setList.count; i++)
        // If command is implemented, set it as PP required.  If the input
        // command is not a PP command, it will be ignored at
```

```
        // PhysicalPresenceCommandSet().
        // Note: PhysicalPresenceCommandSet() checks if the command is implemented.
        PhysicalPresenceCommandSet(in->setList.commandCodes[i]);

    // Process clear list
    for(i = 0; i < in->clearList.count; i++)
        // If command is implemented, clear it as PP required.  If the input
        // command is not a PP command, it will be ignored at
        // PhysicalPresenceCommandClear().  If the input command is
        // TPM2_PP_Commands, it will be ignored as well
        PhysicalPresenceCommandClear(in->clearList.commandCodes[i]);

    // Save the change of PP list
    NV_SYNC_PERSISTENT(ppList);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PP_Commands
```

7.83 /tpm/src/command/Misc/SetAlgorithmSet.c

```
#include "Tpm.h"
#include "SetAlgorithmSet_fp.h"

#if CC_SetAlgorithmSet  // Conditional expansion of this file

/*(See part 3 specification)
// This command allows the platform to change the algorithm set setting of the TPM
*/
TPM_RC
TPM2_SetAlgorithmSet(SetAlgorithmSet_In* in  // IN: input parameter list
)
{
    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Internal Data Update
    gp.algorithmSet = in->algorithmSet;

    // Write the algorithm set changes to NV
    NV_SYNC_PERSISTENT(algorithmSet);

    return TPM_RC_SUCCESS;
}

#endif  // CC_SetAlgorithmSet
```

7.84 /tpm/src/command/NVStorage/NV_Certify.c

```
#include "Tpm.h"
#include "Attest_spt_fp.h"
#include "NV_Certify_fp.h"

#if CC_NV_Certify  // Conditional expansion of this file

/*(See part 3 specification)
// certify the contents of an NV index or portion of an NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_NV_AUTHORIZATION         the authorization was valid but the
//                                      authorizing entity ('authHandle')
```

```
//                                      is not allowed to read from the Index
//                                      referenced by 'nvIndex'
//      TPM_RC_KEY                      'signHandle' does not reference a signing
//                                      key
//      TPM_RC_NV_LOCKED                Index referenced by 'nvIndex' is locked
//                                      for reading
//      TPM_RC_NV_RANGE                 'offset' plus 'size' extends outside of the
//                                      data range of the Index referenced by
//                                      'nvIndex'
//      TPM_RC_NV_UNINITIALIZED         Index referenced by 'nvIndex' has not been
//                                      written
//      TPM_RC_SCHEME                   'inScheme' is not an allowed value for the
//                                      key definition
TPM_RC
TPM2_NV_Certify(NV_Certify_In*  in,  // IN: input parameter list
                NV_Certify_Out* out  // OUT: output parameter list
)
{
    TPM_RC      result;
    NV_REF      locator;
    NV_INDEX*   nvIndex = NvGetIndexInfo(in->nvIndex, &locator);
    TPMS_ATTEST certifyInfo;
    OBJECT*     signObject = HandleToObject(in->signHandle);
    // Input Validation
    if(!IsSigningObject(signObject))
        return TPM_RCS_KEY + RC_NV_Certify_signHandle;
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_NV_Certify_inScheme;

    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    result = NvReadAccessChecks(
        in->authHandle, in->nvIndex, nvIndex->publicArea.attributes);
    if(result != TPM_RC_SUCCESS)
        return result;

    // make sure that the selection is within the range of the Index (cast to avoid
    // any wrap issues with addition)
    if((UINT32)in->size + (UINT32)in->offset > (UINT32)nvIndex->publicArea.dataSize)
        return TPM_RC_NV_RANGE;
    // Make sure the data will fit the return buffer.
    // NOTE: This check may be modified if the output buffer will not hold the
    // maximum sized NV buffer as part of the certified data. The difference in
    // size could be substantial if the signature scheme was produced a large
    // signature (e.g., RSA 4096).
    if(in->size > MAX_NV_BUFFER_SIZE)
        return TPM_RCS_VALUE + RC_NV_Certify_size;

    // Command Output

    // Fill in attest information common fields
    FillInAttestInfo(
        in->signHandle, &in->inScheme, &in->qualifyingData, &certifyInfo);

    // Get the name of the index
    NvGetIndexName(nvIndex, &certifyInfo.attested.nv.indexName);

    // See if this is old format or new format
    if((in->size != 0) || (in->offset != 0))
    {
        // NV certify specific fields
        // Attestation type
        certifyInfo.type = TPM_ST_ATTEST_NV;

        // Set the return size
        certifyInfo.attested.nv.nvContents.t.size = in->size;
```

```
        // Set the offset
        certifyInfo.attested.nv.offset = in->offset;

        // Perform the read
        NvGetIndexData(nvIndex,
                       locator,
                       in->offset,
                       in->size,
                       certifyInfo.attested.nv.nvContents.t.buffer);
    }
    else
    {
        HASH_STATE hashState;
        // This is to sign a digest of the data
        certifyInfo.type = TPM_ST_ATTEST_NV_DIGEST;
        // Initialize the hash before calling the function to add the Index data to
        // the hash.
        certifyInfo.attested.nvDigest.nvDigest.t.size =
            CryptHashStart(&hashState, in->inScheme.details.any.hashAlg);
        NvHashIndexData(
            &hashState, nvIndex, locator, 0, nvIndex->publicArea.dataSize);
        CryptHashEnd2B(&hashState, &certifyInfo.attested.nvDigest.nvDigest.b);
    }
    // Sign attestation structure.  A NULL signature will be returned if
    // signObject is NULL.
    return SignAttestInfo(signObject,
                          &in->inScheme,
                          &certifyInfo,
                          &in->qualifyingData,
                          &out->certifyInfo,
                          &out->signature);
}

#endif  // CC_NV_Certify
```

7.85 /tpm/src/command/NVStorage/NV_ChangeAuth.c

```
#include "Tpm.h"
#include "NV_ChangeAuth_fp.h"

#if CC_NV_ChangeAuth  // Conditional expansion of this file

/*(See part 3 specification)
// change authorization value of a NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_SIZE                     'newAuth' size is larger than the digest
//                                      size of the Name algorithm for the Index
//                                      referenced by 'nvIndex'
TPM_RC
TPM2_NV_ChangeAuth(NV_ChangeAuth_In* in  // IN: input parameter list
)
{
    NV_REF    locator;
    NV_INDEX* nvIndex = NvGetIndexInfo(in->nvIndex, &locator);

    // Input Validation

    // Remove trailing zeros and make sure that the result is not larger than the
    // digest of the nameAlg.
    if(MemoryRemoveTrailingZeros(&in->newAuth)
       > CryptHashGetDigestSize(nvIndex->publicArea.nameAlg))
        return TPM_RCS_SIZE + RC_NV_ChangeAuth_newAuth;
```

```
    // Internal Data Update
    // Change authValue
    return NvWriteIndexAuth(locator, &in->newAuth);
}

#endif  // CC_NV_ChangeAuth
```

7.86 /tpm/src/command/NVStorage/NV_DefineSpace.c

```
#include "Tpm.h"
#include "NV_DefineSpace_fp.h"

#if CC_NV_DefineSpace  // Conditional expansion of this file

/*(See part 3 specification)
// Define a NV index space
*/
//  Return Type: TPM_RC
//      TPM_RC_HIERARCHY            for authorizations using TPM_RH_PLATFORM
//                                  phEnable_NV is clear preventing access to NV
//                                  data in the platform hierarchy.
//      TPM_RC_ATTRIBUTES           attributes of the index are not consistent
//      TPM_RC_NV_DEFINED           index already exists
//      TPM_RC_NV_SPACE             insufficient space for the index
//      TPM_RC_SIZE                 'auth->size' or 'publicInfo->authPolicy.size' is
//                                  larger than the digest size of
//                                  'publicInfo->nameAlg'; or 'publicInfo->dataSize'
//                                  is not consistent with 'publicInfo->attributes'
//                                  (this includes the case when the index is
//                                   larger than a MAX_NV_BUFFER_SIZE but the
//                                   TPMA_NV_WRITEALL attribute is SET)
TPM_RC
TPM2_NV_DefineSpace(NV_DefineSpace_In* in  // IN: input parameter list
)
{
    // This command only supports TPM_HT_NV_INDEX-typed NV indices.
    if(HandleGetType(in->publicInfo.nvPublic.nvIndex) != TPM_HT_NV_INDEX)
    {
        return TPM_RCS_HANDLE + RC_NV_DefineSpace_publicInfo;
    }

    return NvDefineSpace(in->authHandle,
                         &in->auth,
                         &in->publicInfo.nvPublic,
                         RC_NV_DefineSpace_authHandle,
                         RC_NV_DefineSpace_auth,
                         RC_NV_DefineSpace_publicInfo);
}

#endif  // CC_NV_DefineSpace
```

7.87 /tpm/src/command/NVStorage/NV_DefineSpace2.c

```
#include "Tpm.h"
#include "NV_DefineSpace2_fp.h"

#if CC_NV_DefineSpace2  // Conditional expansion of this file

/*(See part 3 specification)
// Define a NV index space
*/
//  Return Type: TPM_RC
//      TPM_RC_HIERARCHY            for authorizations using TPM_RH_PLATFORM
//                                  phEnable_NV is clear preventing access to NV
```

```
//                                  data in the platform hierarchy.
//      TPM_RC_ATTRIBUTES           attributes of the index are not consistent
//      TPM_RC_NV_DEFINED           index already exists
//      TPM_RC_NV_SPACE             insufficient space for the index
//      TPM_RC_SIZE                 'auth->size' or 'publicInfo->authPolicy.size' is
//                                  larger than the digest size of
//                                  'publicInfo->nameAlg'; or 'publicInfo->dataSize'
//                                  is not consistent with 'publicInfo->attributes'
//                                  (this includes the case when the index is
//                                   larger than a MAX_NV_BUFFER_SIZE but the
//                                   TPMA_NV_WRITEALL attribute is SET)
TPM_RC
TPM2_NV_DefineSpace2(NV_DefineSpace2_In* in  // IN: input parameter list
)
{
    TPM_RC         result;
    TPMS_NV_PUBLIC legacyPublic;

    // Input Validation

    // Validate the handle type and the (handle-type-specific) attributes.
    switch(in->publicInfo.nvPublic2.handleType)
    {
        case TPM_HT_NV_INDEX:
            break;
#  if EXTERNAL_NV
        case TPM_HT_EXTERNAL_NV:
            // The reference implementation may let you define an "external" NV
            // index, but it doesn't currently support setting any of the extended
            // bits for customizing the behavior of external NV.
            if((TPMA_NV_EXP_TO_UINT64(
                    in->publicInfo.nvPublic2.nvPublic2.externalNV.attributes)
                & 0xffffffff00000000)
               != 0)
            {
                return TPM_RCS_ATTRIBUTES + RC_NV_DefineSpace2_publicInfo;
            }
            break;
#  endif
        default:
            return TPM_RCS_HANDLE + RC_NV_DefineSpace2_publicInfo;
    }

    result = NvPublicFromNvPublic2(&in->publicInfo.nvPublic2, &legacyPublic);
    if(result != TPM_RC_SUCCESS)
    {
        return RcSafeAddToResult(result, RC_NV_DefineSpace2_publicInfo);
    }

    return NvDefineSpace(in->authHandle,
                         &in->auth,
                         &legacyPublic,
                         RC_NV_DefineSpace2_authHandle,
                         RC_NV_DefineSpace2_auth,
                         RC_NV_DefineSpace2_publicInfo);
}

#endif  // CC_NV_DefineSpace
```

7.88 /tpm/src/command/NVStorage/NV_Extend.c

```
#include "Tpm.h"
#include "NV_Extend_fp.h"

#if CC_NV_Extend  // Conditional expansion of this file
```

```
/*(See part 3 specification)
// Write to a NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               the TPMA_NV_EXTEND attribute is not SET in
//                                      the Index referenced by 'nvIndex'
//      TPM_RC_NV_AUTHORIZATION         the authorization was valid but the
//                                      authorizing entity ('authHandle')
//                                      is not allowed to write to the Index
//                                      referenced by 'nvIndex'
//      TPM_RC_NV_LOCKED                the Index referenced by 'nvIndex' is locked
//                                      for writing
TPM_RC
TPM2_NV_Extend(NV_Extend_In* in  // IN: input parameter list
)
{
    TPM_RC       result;
    NV_REF       locator;
    NV_INDEX*    nvIndex = NvGetIndexInfo(in->nvIndex, &locator);

    TPM2B_DIGEST oldDigest;
    TPM2B_DIGEST newDigest;
    HASH_STATE   hashState;

    // Input Validation

    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    result = NvWriteAccessChecks(
        in->authHandle, in->nvIndex, nvIndex->publicArea.attributes);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Make sure that this is an extend index
    if(!IsNvExtendIndex(nvIndex->publicArea.attributes))
        return TPM_RCS_ATTRIBUTES + RC_NV_Extend_nvIndex;

    // Internal Data Update

    // Perform the write.
    oldDigest.t.size = CryptHashGetDigestSize(nvIndex->publicArea.nameAlg);
    pAssert(oldDigest.t.size <= sizeof(oldDigest.t.buffer));
    if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
    {
        NvGetIndexData(nvIndex, locator, 0, oldDigest.t.size, oldDigest.t.buffer);
    }
    else
    {
        MemorySet(oldDigest.t.buffer, 0, oldDigest.t.size);
    }
    // Start hash
    newDigest.t.size = CryptHashStart(&hashState, nvIndex->publicArea.nameAlg);

    // Adding old digest
    CryptDigestUpdate2B(&hashState, &oldDigest.b);

    // Adding new data
    CryptDigestUpdate2B(&hashState, &in->data.b);

    // Complete hash
    CryptHashEnd2B(&hashState, &newDigest.b);

    // Write extended hash back.
    // Note, this routine will SET the TPMA_NV_WRITTEN attribute if necessary
    return NvWriteIndexData(nvIndex, 0, newDigest.t.size, newDigest.t.buffer);
```

```
}

#endif  // CC_NV_Extend
```

7.89 /tpm/src/command/NVStorage/NV_GlobalWriteLock.c

```
#include "Tpm.h"
#include "NV_GlobalWriteLock_fp.h"

#if CC_NV_GlobalWriteLock  // Conditional expansion of this file

/*(See part 3 specification)
// Set global write lock for NV index
*/
TPM_RC
TPM2_NV_GlobalWriteLock(NV_GlobalWriteLock_In* in  // IN: input parameter list
)
{
    // Input parameter (the authorization handle) is not reference in command action.
    NOT_REFERENCED(in);

    // Internal Data Update

    // Implementation dependent method of setting the global lock
    return NvSetGlobalLock();
}

#endif  // CC_NV_GlobalWriteLock
```

7.90 /tpm/src/command/NVStorage/NV_Increment.c

```
#include "Tpm.h"
#include "NV_Increment_fp.h"

#if CC_NV_Increment  // Conditional expansion of this file

/*(See part 3 specification)
// Increment a NV counter
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               NV index is not a counter
//      TPM_RC_NV_AUTHORIZATION         authorization failure
//      TPM_RC_NV_LOCKED                Index is write locked
TPM_RC
TPM2_NV_Increment(NV_Increment_In* in  // IN: input parameter list
)
{
    TPM_RC    result;
    NV_REF    locator;
    NV_INDEX* nvIndex = NvGetIndexInfo(in->nvIndex, &locator);
    UINT64    countValue;

    // Input Validation

    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    result = NvWriteAccessChecks(
        in->authHandle, in->nvIndex, nvIndex->publicArea.attributes);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Make sure that this is a counter
    if(!IsNvCounterIndex(nvIndex->publicArea.attributes))
        return TPM_RCS_ATTRIBUTES + RC_NV_Increment_nvIndex;
```

```
    // Internal Data Update

    // If counter index is not been written, initialize it
    if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
        countValue = NvReadMaxCount();
    else
        // Read NV data in native format for TPM CPU.
        countValue = NvGetUINT64Data(nvIndex, locator);

    // Do the increment
    countValue++;

    // Write NV data back. A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may
    // be returned at this point. If necessary, this function will set the
    // TPMA_NV_WRITTEN attribute
    result = NvWriteUINT64Data(nvIndex, countValue);
    if(result == TPM_RC_SUCCESS)
    {
        // If a counter just rolled over, then force the NV update.
        // Note, if this is an orderly counter, then the write-back needs to be
        // forced, for other counters, the write-back will happen anyway
        if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, ORDERLY)
           && (countValue & MAX_ORDERLY_COUNT) == 0)
        {
            // Need to force an NV update of orderly data
            SET_NV_UPDATE(UT_ORDERLY);
        }
    }
    return result;
}

#endif  // CC_NV_Increment
```

7.91 /tpm/src/command/NVStorage/NV_Read.c

```
#include "Tpm.h"
#include "NV_Read_fp.h"

#if CC_NV_Read  // Conditional expansion of this file

/*(See part 3 specification)
// Read of an NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_NV_AUTHORIZATION         the authorization was valid but the
//                                      authorizing entity ('authHandle')
//                                      is not allowed to read from the Index
//                                      referenced by 'nvIndex'
//      TPM_RC_NV_LOCKED                the Index referenced by 'nvIndex' is
//                                      read locked
//      TPM_RC_NV_RANGE                 read range defined by 'size' and 'offset'
//                                      is outside the range of the Index referenced
//                                      by 'nvIndex'
//      TPM_RC_NV_UNINITIALIZED         the Index referenced by 'nvIndex' has
//                                      not been initialized (written)
//      TPM_RC_VALUE                    the read size is larger than the
//                                      MAX_NV_BUFFER_SIZE
TPM_RC
TPM2_NV_Read(NV_Read_In*  in,  // IN: input parameter list
             NV_Read_Out* out  // OUT: output parameter list
)
{
    NV_REF    locator;
    NV_INDEX* nvIndex = NvGetIndexInfo(in->nvIndex, &locator);
```

```
    TPM_RC    result;

    // Input Validation
    // Common read access checks. NvReadAccessChecks() may return
    // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
    result = NvReadAccessChecks(
        in->authHandle, in->nvIndex, nvIndex->publicArea.attributes);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Make sure the data will fit the return buffer
    if(in->size > MAX_NV_BUFFER_SIZE)
        return TPM_RCS_VALUE + RC_NV_Read_size;

    // Verify that the offset is not too large
    if(in->offset > nvIndex->publicArea.dataSize)
        return TPM_RCS_VALUE + RC_NV_Read_offset;

    // Make sure that the selection is within the range of the Index
    if(in->size > (nvIndex->publicArea.dataSize - in->offset))
        return TPM_RC_NV_RANGE;

    // Command Output
    // Set the return size
    out->data.t.size = in->size;

    // Perform the read
    NvGetIndexData(nvIndex, locator, in->offset, in->size, out->data.t.buffer);

    return TPM_RC_SUCCESS;
}

#endif  // CC_NV_Read
```

7.92 /tpm/src/command/NVStorage/NV_ReadLock.c

```
#include "Tpm.h"
#include "NV_ReadLock_fp.h"

#if CC_NV_ReadLock  // Conditional expansion of this file

/*(See part 3 specification)
// Set read lock on a NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               TPMA_NV_READ_STCLEAR is not SET so
//                                      Index referenced by 'nvIndex' may not be
//                                      write locked
//      TPM_RC_NV_AUTHORIZATION         the authorization was valid but the
//                                      authorizing entity ('authHandle')
//                                      is not allowed to read from the Index
//                                      referenced by 'nvIndex'
TPM_RC
TPM2_NV_ReadLock(NV_ReadLock_In* in  // IN: input parameter list
)
{
    TPM_RC result;
    NV_REF locator;
    // The referenced index has been checked multiple times before this is called
    // so it must be present and will be loaded into cache
    NV_INDEX* nvIndex      = NvGetIndexInfo(in->nvIndex, &locator);
    TPMA_NV   nvAttributes = nvIndex->publicArea.attributes;

    // Input Validation
    // Common read access checks. NvReadAccessChecks() may return
```

```
    // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
    result = NvReadAccessChecks(in->authHandle, in->nvIndex, nvAttributes);
    if(result == TPM_RC_NV_AUTHORIZATION)
        return TPM_RC_NV_AUTHORIZATION;
    // Index is already locked for write
    else if(result == TPM_RC_NV_LOCKED)
        return TPM_RC_SUCCESS;

    // If NvReadAccessChecks return TPM_RC_NV_UNINITALIZED, then continue.
    // It is not an error to read lock an uninitialized Index.

    // if TPMA_NV_READ_STCLEAR is not set, the index can not be read-locked
    if(!IS_ATTRIBUTE(nvAttributes, TPMA_NV, READ_STCLEAR))
        return TPM_RCS_ATTRIBUTES + RC_NV_ReadLock_nvIndex;

    // Internal Data Update

    // Set the READLOCK attribute
    SET_ATTRIBUTE(nvAttributes, TPMA_NV, READLOCKED);

    // Write NV info back
    return NvWriteIndexAttributes(nvIndex->publicArea.nvIndex, locator, nvAttributes);
}

#endif  // CC_NV_ReadLock
```

7.93 /tpm/src/command/NVStorage/NV_ReadPublic.c

```
#include "Tpm.h"
#include "NV_ReadPublic_fp.h"

#if CC_NV_ReadPublic  // Conditional expansion of this file

/*(See part 3 specification)
// Read the public information of a NV index
*/
TPM_RC
TPM2_NV_ReadPublic(NV_ReadPublic_In*  in,  // IN: input parameter list
                   NV_ReadPublic_Out* out  // OUT: output parameter list
)
{
    NV_INDEX* nvIndex;

    // This command only supports TPM_HT_NV_INDEX-typed NV indices.
    if(HandleGetType(in->nvIndex) != TPM_HT_NV_INDEX)
    {
        return TPM_RCS_HANDLE + RC_NV_ReadPublic_nvIndex;
    }

    nvIndex = NvGetIndexInfo(in->nvIndex, NULL);

    // Command Output

    // Copy index public data to output
    out->nvPublic.nvPublic = nvIndex->publicArea;

    // Compute NV name
    NvGetIndexName(nvIndex, &out->nvName);

    return TPM_RC_SUCCESS;
}

#endif  // CC_NV_ReadPublic
```

7.94 /tpm/src/command/NVStorage/NV_ReadPublic2.c

```
#include "Tpm.h"
#include "NV_ReadPublic2_fp.h"

#if CC_NV_ReadPublic2  // Conditional expansion of this file

/*(See part 3 specification)
// Read the public information of a NV index
*/
TPM_RC
TPM2_NV_ReadPublic2(NV_ReadPublic2_In*  in,  // IN: input parameter list
                    NV_ReadPublic2_Out* out  // OUT: output parameter list
)
{
    TPM_RC    result;
    NV_INDEX* nvIndex;

    nvIndex = NvGetIndexInfo(in->nvIndex, NULL);

    // Command Output

    // The reference code stores its NV indices in the legacy form, because
    // it doesn't support any extended attributes.
    // Translate the legacy form to the general form.
    result = NvPublic2FromNvPublic(&nvIndex->publicArea, &out->nvPublic.nvPublic2);
    if(result != TPM_RC_SUCCESS)
    {
        return RcSafeAddToResult(result, RC_NV_ReadPublic2_nvIndex);
    }

    // Compute NV name
    NvGetIndexName(nvIndex, &out->nvName);

    return TPM_RC_SUCCESS;
}

#endif  // CC_NV_ReadPublic2
```

7.95 /tpm/src/command/NVStorage/NV_SetBits.c

```
#include "Tpm.h"
#include "NV_SetBits_fp.h"

#if CC_NV_SetBits  // Conditional expansion of this file

/*(See part 3 specification)
// Set bits in a NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               the TPMA_NV_BITS attribute is not SET in the
//                                      Index referenced by 'nvIndex'
//      TPM_RC_NV_AUTHORIZATION         the authorization was valid but the
//                                      authorizing entity ('authHandle')
//                                      is not allowed to write to the Index
//                                      referenced by 'nvIndex'
//      TPM_RC_NV_LOCKED                the Index referenced by 'nvIndex' is locked
//                                      for writing
TPM_RC
TPM2_NV_SetBits(NV_SetBits_In* in  // IN: input parameter list
)
{
    TPM_RC    result;
    NV_REF    locator;
    NV_INDEX* nvIndex = NvGetIndexInfo(in->nvIndex, &locator);
```

```
    UINT64    oldValue;
    UINT64    newValue;

    // Input Validation

    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    result = NvWriteAccessChecks(
        in->authHandle, in->nvIndex, nvIndex->publicArea.attributes);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Make sure that this is a bit field
    if(!IsNvBitsIndex(nvIndex->publicArea.attributes))
        return TPM_RCS_ATTRIBUTES + RC_NV_SetBits_nvIndex;

    // If index is not been written, initialize it
    if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
        oldValue = 0;
    else
        // Read index data
        oldValue = NvGetUINT64Data(nvIndex, locator);

    // Figure out what the new value is going to be
    newValue = oldValue | in->bits;

    // Internal Data Update
    return NvWriteUINT64Data(nvIndex, newValue);
}

#endif  // CC_NV_SetBits
```

7.96 /tpm/src/command/NVStorage/NV_spt.c

```
//** Includes
#include "Tpm.h"
#include "NV_spt_fp.h"

//** Functions

//*** NvReadAccessChecks()
// Common routine for validating a read
// Used by TPM2_NV_Read, TPM2_NV_ReadLock and TPM2_PolicyNV
//  Return Type: TPM_RC
//      TPM_RC_NV_AUTHORIZATION     authHandle is not allowed to authorize read
//                                  of the index
//      TPM_RC_NV_LOCKED            Read locked
//      TPM_RC_NV_UNINITIALIZED     Try to read an uninitialized index
//
TPM_RC
NvReadAccessChecks(TPM_HANDLE authHandle,  // IN: the handle that provided the
                                           //     authorization
                   TPM_HANDLE nvHandle,   // IN: the handle of the NV index to be read
                   TPMA_NV    attributes  // IN: the attributes of 'nvHandle'
)
{
    // If data is read locked, returns an error
    if(IS_ATTRIBUTE(attributes, TPMA_NV, READLOCKED))
        return TPM_RC_NV_LOCKED;
    // If the authorization was provided by the owner or platform, then check
    // that the attributes allow the read.  If the authorization handle
    // is the same as the index, then the checks were made when the authorization
    // was checked..
    if(authHandle == TPM_RH_OWNER)
    {
```

```
        // If Owner provided authorization then ONWERWRITE must be SET
        if(!IS_ATTRIBUTE(attributes, TPMA_NV, OWNERREAD))
            return TPM_RC_NV_AUTHORIZATION;
    }
    else if(authHandle == TPM_RH_PLATFORM)
    {
        // If Platform provided authorization then PPWRITE must be SET
        if(!IS_ATTRIBUTE(attributes, TPMA_NV, PPREAD))
            return TPM_RC_NV_AUTHORIZATION;
    }
    // If neither Owner nor Platform provided authorization, make sure that it was
    // provided by this index.
    else if(authHandle != nvHandle)
        return TPM_RC_NV_AUTHORIZATION;

    // If the index has not been written, then the value cannot be read
    // NOTE: This has to come after other access checks to make sure that
    // the proper authorization is given to TPM2_NV_ReadLock()
    if(!IS_ATTRIBUTE(attributes, TPMA_NV, WRITTEN))
        return TPM_RC_NV_UNINITIALIZED;

    return TPM_RC_SUCCESS;
}

//*** NvWriteAccessChecks()
// Common routine for validating a write
// Used by TPM2_NV_Write, TPM2_NV_Increment, TPM2_SetBits, and TPM2_NV_WriteLock
//  Return Type: TPM_RC
//      TPM_RC_NV_AUTHORIZATION     Authorization fails
//      TPM_RC_NV_LOCKED            Write locked
//
TPM_RC
NvWriteAccessChecks(
    TPM_HANDLE authHandle,  // IN: the handle that provided the
                            //     authorization
    TPM_HANDLE nvHandle,    // IN: the handle of the NV index to be written
    TPMA_NV    attributes   // IN: the attributes of 'nvHandle'
)
{
    // If data is write locked, returns an error
    if(IS_ATTRIBUTE(attributes, TPMA_NV, WRITELOCKED))
        return TPM_RC_NV_LOCKED;
    // If the authorization was provided by the owner or platform, then check
    // that the attributes allow the write.  If the authorization handle
    // is the same as the index, then the checks were made when the authorization
    // was checked..
    if(authHandle == TPM_RH_OWNER)
    {
        // If Owner provided authorization then ONWERWRITE must be SET
        if(!IS_ATTRIBUTE(attributes, TPMA_NV, OWNERWRITE))
            return TPM_RC_NV_AUTHORIZATION;
    }
    else if(authHandle == TPM_RH_PLATFORM)
    {
        // If Platform provided authorization then PPWRITE must be SET
        if(!IS_ATTRIBUTE(attributes, TPMA_NV, PPWRITE))
            return TPM_RC_NV_AUTHORIZATION;
    }
    // If neither Owner nor Platform provided authorization, make sure that it was
    // provided by this index.
    else if(authHandle != nvHandle)
        return TPM_RC_NV_AUTHORIZATION;
    return TPM_RC_SUCCESS;
}

//*** NvClearOrderly()
```

```
// This function is used to cause gp.orderlyState to be cleared to the
// non-orderly state.
TPM_RC
NvClearOrderly(void)
{
    if(gp.orderlyState < SU_DA_USED_VALUE)
        RETURN_IF_NV_IS_NOT_AVAILABLE;
    g_clearOrderly = TRUE;
    return TPM_RC_SUCCESS;
}

//*** NvIsPinPassIndex()
// Function to check to see if an NV index is a PIN Pass Index
//  Return Type: BOOL
//      TRUE(1)         is pin pass
//      FALSE(0)        is not pin pass
BOOL NvIsPinPassIndex(TPM_HANDLE index  // IN: Handle to check
)
{
    if(HandleGetType(index) == TPM_HT_NV_INDEX)
    {
        NV_INDEX* nvIndex = NvGetIndexInfo(index, NULL);

        return IsNvPinPassIndex(nvIndex->publicArea.attributes);
    }
    return FALSE;
}

//*** NvGetIndexName()
// This function computes the Name of an index
// The 'name' buffer receives the bytes of the Name and the return value
// is the number of octets in the Name.
//
// This function requires that the NV Index is defined.
TPM2B_NAME* NvGetIndexName(
    NV_INDEX* nvIndex,  // IN: the index over which the name is to be
                        //     computed
    TPM2B_NAME* name    // OUT: name of the index
)
{
    UINT16           dataSize, digestSize;
    BYTE             marshalBuffer[sizeof(TPMU_NV_PUBLIC_2)];
    BYTE*            buffer;
    INT32            bufferSize = sizeof(marshalBuffer);
    HASH_STATE       hashState;
    TPMT_NV_PUBLIC_2 public2;

    // Convert the legacy representation into the tagged-union representation.
    NvPublic2FromNvPublic(&nvIndex->publicArea, &public2);

    // Marshal the whole public area, but not the TPM_HT selector:
    // This is safe, because the TPM_HT is the first byte of the handle value,
    // which is already in every element of TPMT_NV_PUBLIC_2.
    // This allows the Name of an NV index calculated based on the
    // TPMT_NV_PUBLIC_2 to be consistent with the Name of the same index if it
    // has a TPMS_NV_PUBLIC representation.
    buffer = marshalBuffer;
    dataSize =
        TPMU_NV_PUBLIC_2_Marshal(&public2.nvPublic2,
                                 &buffer,
                                 &bufferSize,
                                 (UINT32)HandleGetType(nvIndex->publicArea.nvIndex));

    // hash public area
    digestSize = CryptHashStart(&hashState, nvIndex->publicArea.nameAlg);
    CryptDigestUpdate(&hashState, dataSize, marshalBuffer);
```

```
    // Complete digest leaving room for the nameAlg
    CryptHashEnd(&hashState, digestSize, &name->b.buffer[2]);

    // Include the nameAlg
    UINT16_TO_BYTE_ARRAY(nvIndex->publicArea.nameAlg, name->b.buffer);
    name->t.size = digestSize + 2;
    return name;
}

// NOTE: This is a lossy conversion: any expanded attributes are lost here.
// Calling code should return an error to the user, instead of dropping their
// data, if any of the expanded attributes are SET.
static TPMA_NV LegacyAttributesFromExpanded(TPMA_NV_EXP attributes)
{
    UINT64 attributes64;
    UINT32 attributes32;

    attributes64 = TPMA_NV_EXP_TO_UINT64(attributes);
    attributes32 = (UINT32)attributes64;

    return UINT32_TO_TPMA_NV(attributes32);
}

static TPMA_NV_EXP ExpandedAttributesFromLegacy(TPMA_NV attributes)
{
    UINT32 attributes32;
    UINT64 attributes64;

    attributes32 = TPMA_NV_TO_UINT32(attributes);
    attributes64 = (UINT64)attributes32;

    return UINT64_TO_TPMA_NV_EXP(attributes64);
}

//*** NvPublic2FromNvPublic()
// This function converts a legacy-form NV public (TPMS_NV_PUBLIC) into the
// generalized TPMT_NV_PUBLIC_2 tagged-union representation.
TPM_RC NvPublic2FromNvPublic(
    TPMS_NV_PUBLIC*   nvPublic,  // IN: the source S-form NV public area
    TPMT_NV_PUBLIC_2* nvPublic2  // OUT: the T-form NV public area to populate
)
{
    TPM_HT handleType = HandleGetType(nvPublic->nvIndex);

    switch(handleType)
    {
        case TPM_HT_NV_INDEX:
            nvPublic2->nvPublic2.nvIndex = *nvPublic;
            break;
        case TPM_HT_PERMANENT_NV:
            nvPublic2->nvPublic2.permanentNV = *nvPublic;
            break;
#if EXTERNAL_NV
        case TPM_HT_EXTERNAL_NV:
        {
            TPMS_NV_PUBLIC_EXP_ATTR* pub = &nvPublic2->nvPublic2.externalNV;

            pub->attributes = ExpandedAttributesFromLegacy(nvPublic->attributes);
            pub->authPolicy = nvPublic->authPolicy;
            pub->dataSize   = nvPublic->dataSize;
            pub->nameAlg    = nvPublic->nameAlg;
            pub->nvIndex    = nvPublic->nvIndex;
            break;
        }
#endif
```

```
        default:
            return TPM_RCS_HANDLE;
    }

    nvPublic2->handleType = handleType;
    return TPM_RC_SUCCESS;
}

//*** NvPublicFromNvPublic2()
// This function converts a tagged-union NV public (TPMT_NV_PUBLIC_2) into the
// legacy TPMS_NV_PUBLIC representation. This is a lossy conversion: any
// bits in the extended area of the attributes are lost, and the Name cannot be
// computed based on it.
TPM_RC NvPublicFromNvPublic2(
    TPMT_NV_PUBLIC_2* nvPublic2,  // IN: the source T-form NV public area
    TPMS_NV_PUBLIC*   nvPublic    // OUT: the S-form NV public area to populate
)
{
    switch(nvPublic2->handleType)
    {
        case TPM_HT_NV_INDEX:
            *nvPublic = nvPublic2->nvPublic2.nvIndex;
            break;
        case TPM_HT_PERMANENT_NV:
            *nvPublic = nvPublic2->nvPublic2.permanentNV;
            break;
#if EXTERNAL_NV
        case TPM_HT_EXTERNAL_NV:
        {
            TPMS_NV_PUBLIC_EXP_ATTR* pub = &nvPublic2->nvPublic2.externalNV;

            nvPublic->attributes = LegacyAttributesFromExpanded(pub->attributes);
            nvPublic->authPolicy = pub->authPolicy;
            nvPublic->dataSize   = pub->dataSize;
            nvPublic->nameAlg    = pub->nameAlg;
            break;
        }
#endif
        default:
            return TPM_RCS_HANDLE;
    }

    return TPM_RC_SUCCESS;
}

//*** NvDefineSpace()
// This function combines the common functionality of TPM2_NV_DefineSpace and
// TPM2_NV_DefineSpace2.
TPM_RC NvDefineSpace(TPMI_RH_PROVISION authHandle,
                     TPM2B_AUTH*       auth,
                     TPMS_NV_PUBLIC*   publicInfo,
                     TPM_RC            blameAuthHandle,
                     TPM_RC            blameAuth,
                     TPM_RC            blamePublic)
{
    TPMA_NV attributes = publicInfo->attributes;
    UINT16  nameSize;

    nameSize = CryptHashGetDigestSize(publicInfo->nameAlg);

    // Input Validation

    // Checks not specific to type

    // If the UndefineSpaceSpecial command is not implemented, then can't have
    // an index that can only be deleted with policy
```

```
#if CC_NV_UndefineSpaceSpecial == NO
    if(IS_ATTRIBUTE(attributes, TPMA_NV, POLICY_DELETE))
        return TPM_RCS_ATTRIBUTES + blamePublic;
#endif

    // check that the authPolicy consistent with hash algorithm

    if(publicInfo->authPolicy.t.size != 0
       && publicInfo->authPolicy.t.size != nameSize)
        return TPM_RCS_SIZE + blamePublic;

    // make sure that the authValue is not too large
    if(MemoryRemoveTrailingZeros(auth) > CryptHashGetDigestSize(publicInfo->nameAlg))
        return TPM_RCS_SIZE + blameAuth;

    // If an index is being created by the owner and shEnable is
    // clear, then we would not reach this point because ownerAuth
    // can't be given when shEnable is CLEAR. However, if phEnable
    // is SET but phEnableNV is CLEAR, we have to check here
    if(authHandle == TPM_RH_PLATFORM && gc.phEnableNV == CLEAR)
        return TPM_RCS_HIERARCHY + blameAuthHandle;

    // Attribute checks
    // Eliminate the unsupported types
    switch(GET_TPM_NT(attributes))
    {
#if CC_NV_Increment == YES
        case TPM_NT_COUNTER:
#endif
#if CC_NV_SetBits == YES
        case TPM_NT_BITS:
#endif
#if CC_NV_Extend == YES
        case TPM_NT_EXTEND:
#endif
#if CC_PolicySecret == YES && defined TPM_NT_PIN_PASS
        case TPM_NT_PIN_PASS:
        case TPM_NT_PIN_FAIL:
#endif
        case TPM_NT_ORDINARY:
            break;
        default:
            return TPM_RCS_ATTRIBUTES + blamePublic;
            break;
    }
    // Check that the sizes are OK based on the type
    switch(GET_TPM_NT(attributes))
    {
        case TPM_NT_ORDINARY:
            // Can't exceed the allowed size for the implementation
            if(publicInfo->dataSize > MAX_NV_INDEX_SIZE)
                return TPM_RCS_SIZE + blamePublic;
            break;
        case TPM_NT_EXTEND:
            if(publicInfo->dataSize != nameSize)
                return TPM_RCS_SIZE + blamePublic;
            break;
        default:
            // Everything else needs a size of 8
            if(publicInfo->dataSize != 8)
                return TPM_RCS_SIZE + blamePublic;
            break;
    }
    // Handle other specifics
    switch(GET_TPM_NT(attributes))
    {
```

```
        case TPM_NT_COUNTER:
            // Counter can't have TPMA_NV_CLEAR_STCLEAR SET (don't clear counters)
            if(IS_ATTRIBUTE(attributes, TPMA_NV, CLEAR_STCLEAR))
                return TPM_RCS_ATTRIBUTES + blamePublic;
            break;
#ifdef TPM_NT_PIN_FAIL
        case TPM_NT_PIN_FAIL:
            // NV_NO_DA must be SET and AUTHWRITE must be CLEAR
            // NOTE: As with a PIN_PASS index, the authValue of the index is not
            // available until the index is written. If AUTHWRITE is the only way to
            // write then index, it could never be written. Rather than go through
            // all of the other possible ways to write the Index, it is simply
            // prohibited to write the index with the authValue. Other checks
            // below will insure that there seems to be a way to write the index
            // (i.e., with platform authorization , owner authorization,
            // or with policyAuth.)
            // It is not allowed to create a PIN Index that can't be modified.
            if(!IS_ATTRIBUTE(attributes, TPMA_NV, NO_DA))
                return TPM_RCS_ATTRIBUTES + blamePublic;
#endif
#ifdef TPM_NT_PIN_PASS
        case TPM_NT_PIN_PASS:
            // AUTHWRITE must be CLEAR (see note above to TPM_NT_PIN_FAIL)
            if(IS_ATTRIBUTE(attributes, TPMA_NV, AUTHWRITE)
               || IS_ATTRIBUTE(attributes, TPMA_NV, GLOBALLOCK)
               || IS_ATTRIBUTE(attributes, TPMA_NV, WRITEDEFINE))
                return TPM_RCS_ATTRIBUTES + blamePublic;
#endif  // this comes before break because PIN_FAIL falls through
            break;
        default:
            break;
    }

    // Locks may not be SET and written cannot be SET
    if(IS_ATTRIBUTE(attributes, TPMA_NV, WRITTEN)
       || IS_ATTRIBUTE(attributes, TPMA_NV, WRITELOCKED)
       || IS_ATTRIBUTE(attributes, TPMA_NV, READLOCKED))
        return TPM_RCS_ATTRIBUTES + blamePublic;

    // There must be a way to read the index.
    if(!IS_ATTRIBUTE(attributes, TPMA_NV, OWNERREAD)
       && !IS_ATTRIBUTE(attributes, TPMA_NV, PPREAD)
       && !IS_ATTRIBUTE(attributes, TPMA_NV, AUTHREAD)
       && !IS_ATTRIBUTE(attributes, TPMA_NV, POLICYREAD))
        return TPM_RCS_ATTRIBUTES + blamePublic;

    // There must be a way to write the index
    if(!IS_ATTRIBUTE(attributes, TPMA_NV, OWNERWRITE)
       && !IS_ATTRIBUTE(attributes, TPMA_NV, PPWRITE)
       && !IS_ATTRIBUTE(attributes, TPMA_NV, AUTHWRITE)
       && !IS_ATTRIBUTE(attributes, TPMA_NV, POLICYWRITE))
        return TPM_RCS_ATTRIBUTES + blamePublic;

    // An index with TPMA_NV_CLEAR_STCLEAR can't have TPMA_NV_WRITEDEFINE SET
    if(IS_ATTRIBUTE(attributes, TPMA_NV, CLEAR_STCLEAR)
       && IS_ATTRIBUTE(attributes, TPMA_NV, WRITEDEFINE))
        return TPM_RCS_ATTRIBUTES + blamePublic;

    // Make sure that the creator of the index can delete the index
    if((IS_ATTRIBUTE(attributes, TPMA_NV, PLATFORMCREATE)
        && authHandle == TPM_RH_OWNER)
       || (!IS_ATTRIBUTE(attributes, TPMA_NV, PLATFORMCREATE)
           && authHandle == TPM_RH_PLATFORM))
        return TPM_RCS_ATTRIBUTES + blameAuthHandle;

    // If TPMA_NV_POLICY_DELETE is SET, then the index must be defined by
```

```
    // the platform
    if(IS_ATTRIBUTE(attributes, TPMA_NV, POLICY_DELETE)
       && TPM_RH_PLATFORM != authHandle)
        return TPM_RCS_ATTRIBUTES + blamePublic;

    // Make sure that the TPMA_NV_WRITEALL is not set if the index size is larger
    // than the allowed NV buffer size.
    if(publicInfo->dataSize > MAX_NV_BUFFER_SIZE
       && IS_ATTRIBUTE(attributes, TPMA_NV, WRITEALL))
        return TPM_RCS_SIZE + blamePublic;

    // And finally, see if the index is already defined.
    if(NvIndexIsDefined(publicInfo->nvIndex))
        return TPM_RC_NV_DEFINED;

    // Internal Data Update
    // define the space.  A TPM_RC_NV_SPACE error may be returned at this point
    return NvDefineIndex(publicInfo, auth);
}
```

7.97 /tpm/src/command/NVStorage/NV_UndefineSpace.c

```
#include "Tpm.h"
#include "NV_UndefineSpace_fp.h"

#if CC_NV_UndefineSpace  // Conditional expansion of this file

/*(See part 3 specification)
// Delete an NV Index
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               TPMA_NV_POLICY_DELETE is SET in the Index
//                                      referenced by 'nvIndex' so this command may
//                                      not be used to delete this Index (see
//                                      TPM2_NV_UndefineSpaceSpecial())
//      TPM_RC_NV_AUTHORIZATION         attempt to use ownerAuth to delete an index
//                                      created by the platform
//
TPM_RC
TPM2_NV_UndefineSpace(NV_UndefineSpace_In* in  // IN: input parameter list
)
{
    NV_REF    locator;
    NV_INDEX* nvIndex = NvGetIndexInfo(in->nvIndex, &locator);

    // Input Validation
    // This command can't be used to delete an index with TPMA_NV_POLICY_DELETE SET
    if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, POLICY_DELETE))
        return TPM_RCS_ATTRIBUTES + RC_NV_UndefineSpace_nvIndex;

    // The owner may only delete an index that was defined with ownerAuth. The
    // platform may delete an index that was created with either authorization.
    if(in->authHandle == TPM_RH_OWNER
       && IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, PLATFORMCREATE))
        return TPM_RC_NV_AUTHORIZATION;

    // Internal Data Update

    // Call implementation dependent internal routine to delete NV index
    return NvDeleteIndex(nvIndex, locator);
}

#endif  // CC_NV_UndefineSpace
```

7.98 /tpm/src/command/NVStorage/NV_UndefineSpaceSpecial.c

```
#include "Tpm.h"
#include "NV_UndefineSpaceSpecial_fp.h"
#include "SessionProcess_fp.h"

#if CC_NV_UndefineSpaceSpecial  // Conditional expansion of this file

/*(See part 3 specification)
// Delete a NV index that requires policy to delete.
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               TPMA_NV_POLICY_DELETE is not SET in the
//                                      Index referenced by 'nvIndex'
TPM_RC
TPM2_NV_UndefineSpaceSpecial(
    NV_UndefineSpaceSpecial_In* in  // IN: input parameter list
)
{
    TPM_RC    result;
    NV_REF    locator;
    NV_INDEX* nvIndex = NvGetIndexInfo(in->nvIndex, &locator);
    // Input Validation
    // This operation only applies when the TPMA_NV_POLICY_DELETE attribute is SET
    if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, POLICY_DELETE))
        return TPM_RCS_ATTRIBUTES + RC_NV_UndefineSpaceSpecial_nvIndex;
    // Internal Data Update
    // Call implementation dependent internal routine to delete NV index
    result = NvDeleteIndex(nvIndex, locator);

    // If we just removed the index providing the authorization, make sure that the
    // authorization session computation is modified so that it doesn't try to
    // access the authValue of the just deleted index
    if(result == TPM_RC_SUCCESS)
        SessionRemoveAssociationToHandle(in->nvIndex);
    return result;
}

#endif  // CC_NV_UndefineSpaceSpecial
```

7.99 /tpm/src/command/NVStorage/NV_Write.c

```
#include "Tpm.h"
#include "NV_Write_fp.h"

#if CC_NV_Write  // Conditional expansion of this file

/*(See part 3 specification)
// Write to a NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               Index referenced by 'nvIndex' has either
//                                      TPMA_NV_BITS, TPMA_NV_COUNTER, or
//                                      TPMA_NV_EVENT attribute SET
//      TPM_RC_NV_AUTHORIZATION         the authorization was valid but the
//                                      authorizing entity ('authHandle')
//                                      is not allowed to write to the Index
//                                      referenced by 'nvIndex'
//      TPM_RC_NV_LOCKED                Index referenced by 'nvIndex' is write
//                                      locked
//      TPM_RC_NV_RANGE                 if TPMA_NV_WRITEALL is SET then the write
//                                      is not the size of the Index referenced by
//                                      'nvIndex'; otherwise, the write extends
//                                      beyond the limits of the Index
//
```

```
TPM_RC
TPM2_NV_Write(NV_Write_In* in  // IN: input parameter list
)
{
    NV_INDEX* nvIndex    = NvGetIndexInfo(in->nvIndex, NULL);
    TPMA_NV   attributes = nvIndex->publicArea.attributes;
    TPM_RC    result;

    // Input Validation

    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    result = NvWriteAccessChecks(in->authHandle, in->nvIndex, attributes);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Bits index, extend index or counter index may not be updated by
    // TPM2_NV_Write
    if(IsNvCounterIndex(attributes) || IsNvBitsIndex(attributes)
       || IsNvExtendIndex(attributes))
        return TPM_RC_ATTRIBUTES;

    // Make sure that the offset is not too large
    if(in->offset > nvIndex->publicArea.dataSize)
        return TPM_RCS_VALUE + RC_NV_Write_offset;

    // Make sure that the selection is within the range of the Index
    if(in->data.t.size > (nvIndex->publicArea.dataSize - in->offset))
        return TPM_RC_NV_RANGE;

    // If this index requires a full sized write, make sure that input range is
    // full sized.
    // Note: if the requested size is the same as the Index data size, then offset
    // will have to be zero. Otherwise, the range check above would have failed.
    if(IS_ATTRIBUTE(attributes, TPMA_NV, WRITEALL)
       && in->data.t.size < nvIndex->publicArea.dataSize)
        return TPM_RC_NV_RANGE;

    // Internal Data Update

    // Perform the write.  This called routine will SET the TPMA_NV_WRITTEN
    // attribute if it has not already been SET. If NV isn't available, an error
    // will be returned.
    return NvWriteIndexData(nvIndex, in->offset, in->data.t.size, in->data.t.buffer);
}

#endif  // CC_NV_Write
```

7.100 /tpm/src/command/NVStorage/NV_WriteLock.c

```
#include "Tpm.h"
#include "NV_WriteLock_fp.h"

#if CC_NV_WriteLock  // Conditional expansion of this file

/*(See part 3 specification)
// Set write lock on a NV index
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES               neither TPMA_NV_WRITEDEFINE nor
//                                      TPMA_NV_WRITE_STCLEAR is SET in Index
//                                      referenced by 'nvIndex'
//      TPM_RC_NV_AUTHORIZATION         the authorization was valid but the
//                                      authorizing entity ('authHandle')
//                                      is not allowed to write to the Index
```

```
//                                      referenced by 'nvIndex'
//
TPM_RC
TPM2_NV_WriteLock(NV_WriteLock_In* in  // IN: input parameter list
)
{
    TPM_RC    result;
    NV_REF    locator;
    NV_INDEX* nvIndex      = NvGetIndexInfo(in->nvIndex, &locator);
    TPMA_NV   nvAttributes = nvIndex->publicArea.attributes;

    // Input Validation:

    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    result = NvWriteAccessChecks(in->authHandle, in->nvIndex, nvAttributes);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_NV_AUTHORIZATION)
            return result;
        // If write access failed because the index is already locked, then it is
        // no error.
        return TPM_RC_SUCCESS;
    }
    // if neither TPMA_NV_WRITEDEFINE nor TPMA_NV_WRITE_STCLEAR is set, the index
    // can not be write-locked
    if(!IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITEDEFINE)
       && !IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITE_STCLEAR))
        return TPM_RCS_ATTRIBUTES + RC_NV_WriteLock_nvIndex;
    // Internal Data Update
    // Set the WRITELOCK attribute.
    // Note: if TPMA_NV_WRITELOCKED were already SET, then the write access check
    // above would have failed and this code isn't executed.
    SET_ATTRIBUTE(nvAttributes, TPMA_NV, WRITELOCKED);

    // Write index info back
    return NvWriteIndexAttributes(nvIndex->publicArea.nvIndex, locator, nvAttributes);
}

#endif  // CC_NV_WriteLock
```

7.101 /tpm/src/command/Object/ActivateCredential.c

```
#include "Tpm.h"
#include "ActivateCredential_fp.h"

#if CC_ActivateCredential  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// Activate Credential with an object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'keyHandle' does not reference a decryption key
//      TPM_RC_ECC_POINT        'secret' is invalid (when 'keyHandle' is an ECC key)
//      TPM_RC_INSUFFICIENT     'secret' is invalid (when 'keyHandle' is an ECC key)
//      TPM_RC_INTEGRITY        'credentialBlob' fails integrity test
//      TPM_RC_NO_RESULT        'secret' is invalid (when 'keyHandle' is an ECC key)
//      TPM_RC_SIZE             'secret' size is invalid or the 'credentialBlob'
//                              does not unmarshal correctly
//      TPM_RC_TYPE             'keyHandle' does not reference an asymmetric key.
//      TPM_RC_VALUE            'secret' is invalid (when 'keyHandle' is an RSA key)
TPM_RC
TPM2_ActivateCredential(ActivateCredential_In*  in,  // IN: input parameter list
```

```
                        ActivateCredential_Out* out  // OUT: output parameter list
)
{
    TPM_RC     result = TPM_RC_SUCCESS;
    OBJECT*    object;          // decrypt key
    OBJECT*    activateObject;  // key associated with credential
    TPM2B_DATA data;            // credential data

    // Input Validation

    // Get decrypt key pointer
    object = HandleToObject(in->keyHandle);

    // Get certificated object pointer
    activateObject = HandleToObject(in->activateHandle);

    // input decrypt key must be an asymmetric, restricted decryption key
    if(!CryptIsAsymAlgorithm(object->publicArea.type)
       || !IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, decrypt)
       || !IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, restricted))
        return TPM_RCS_TYPE + RC_ActivateCredential_keyHandle;

    // Command output

    // Decrypt input credential data via asymmetric decryption.  A
    // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
    // point
    result = CryptSecretDecrypt(object, NULL, IDENTITY_STRING, &in->secret, &data);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_FAILURE;
        return RcSafeAddToResult(result, RC_ActivateCredential_secret);
    }

    // Retrieve secret data.  A TPM_RC_INTEGRITY error or unmarshal
    // errors may be returned at this point
    result = CredentialToSecret(&in->credentialBlob.b,
                                &activateObject->name.b,
                                &data.b,
                                object,
                                &out->certInfo);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_ActivateCredential_credentialBlob);

    return TPM_RC_SUCCESS;
}

#endif  // CC_ActivateCredential
```

7.102 /tpm/src/command/Object/Create.c

```
#include "Tpm.h"
#include "Object_spt_fp.h"
#include "Create_fp.h"

#if CC_Create  // Conditional expansion of this file

/*(See part 3 specification)
// Create a regular object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'sensitiveDataOrigin' is CLEAR when 'sensitive.data'
//                              is an Empty Buffer, or is SET when 'sensitive.data' is
//                              not empty;
```

```
//                              'fixedTPM', 'fixedParent', or 'encryptedDuplication'
//                              attributes are inconsistent between themselves or with
//                              those of the parent object;
//                              inconsistent 'restricted', 'decrypt' and 'sign'
//                              attributes;
//                              attempt to inject sensitive data for an asymmetric
//                              key;
//      TPM_RC_HASH             non-duplicable storage key and its parent have
//                              different name algorithm
//      TPM_RC_KDF              incorrect KDF specified for decrypting keyed hash
//                              object
//      TPM_RC_KEY              invalid key size values in an asymmetric key public
//                              area or a provided symmetric key has a value that is
//                              not allowed
//      TPM_RC_KEY_SIZE         key size in public area for symmetric key differs from
//                              the size in the sensitive creation area; may also be
//                              returned if the TPM does not allow the key size to be
//                              used for a Storage Key
//      TPM_RC_OBJECT_MEMORY    a free slot is not available as scratch memory for
//                              object creation
//      TPM_RC_RANGE            the exponent value of an RSA key is not supported.
//      TPM_RC_SCHEME           inconsistent attributes 'decrypt', 'sign', or
//                              'restricted' and key's scheme ID; or hash algorithm is
//                              inconsistent with the scheme ID for keyed hash object
//      TPM_RC_SIZE             size of public authPolicy or sensitive authValue does
//                              not match digest size of the name algorithm
//                              sensitive data size for the keyed hash object is
//                              larger than is allowed for the scheme
//      TPM_RC_SYMMETRIC        a storage key with no symmetric algorithm specified;
//                              or non-storage key with symmetric algorithm different
//                              from TPM_ALG_NULL
//      TPM_RC_TYPE             unknown object type;
//                              'parentHandle' does not reference a restricted
//                              decryption key in the storage hierarchy with both
//                              public and sensitive portion loaded
//      TPM_RC_VALUE            exponent is not prime or could not find a prime using
//                              the provided parameters for an RSA key;
//                              unsupported name algorithm for an ECC key
//      TPM_RC_OBJECT_MEMORY    there is no free slot for the object
TPM_RC
TPM2_Create(Create_In*  in,  // IN: input parameter list
            Create_Out* out  // OUT: output parameter list
)
{
    TPM_RC       result = TPM_RC_SUCCESS;
    OBJECT*      parentObject;
    OBJECT*      newObject;
    TPMT_PUBLIC* publicArea;

    // Input Validation
    parentObject = HandleToObject(in->parentHandle);
    pAssert(parentObject != NULL);

    // Does parent have the proper attributes?
    if(!ObjectIsParent(parentObject))
        return TPM_RCS_TYPE + RC_Create_parentHandle;

    // Get a slot for the creation
    newObject = FindEmptyObjectSlot(NULL);
    if(newObject == NULL)
        return TPM_RC_OBJECT_MEMORY;
    // If the TPM2B_PUBLIC was passed as a structure, marshal it into is canonical
    // form for processing

    // to save typing.
    publicArea = &newObject->publicArea;
```

```
    // Copy the input structure to the allocated structure
    *publicArea = in->inPublic.publicArea;

    // Check attributes in input public area. CreateChecks() checks the things that
    // are unique to creation and then validates the attributes and values that are
    // common to create and load.
    result = CreateChecks(parentObject,
                          /* primaryHierarchy = */ 0,
                          publicArea,
                          in->inSensitive.sensitive.data.t.size);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_Create_inPublic);
    // Clean up the authValue if necessary
    if(!AdjustAuthSize(&in->inSensitive.sensitive.userAuth, publicArea->nameAlg))
        return TPM_RCS_SIZE + RC_Create_inSensitive;

    // Command Output
    // Create the object using the default TPM random-number generator
    result = CryptCreateObject(newObject, &in->inSensitive.sensitive, NULL);
    if(result != TPM_RC_SUCCESS)
        return result;
    // Fill in creation data
    FillInCreationData(in->parentHandle,
                       publicArea->nameAlg,
                       &in->creationPCR,
                       &in->outsideInfo,
                       &out->creationData,
                       &out->creationHash);

    // Compute creation ticket
    result = TicketComputeCreation(EntityGetHierarchy(in->parentHandle),
                                   &newObject->name,
                                   &out->creationHash,
                                   &out->creationTicket);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Prepare output private data from sensitive
    SensitiveToPrivate(&newObject->sensitive,
                       &newObject->name,
                       parentObject,
                       publicArea->nameAlg,
                       &out->outPrivate);

    newObject->hierarchy = parentObject->hierarchy;

    // Finish by copying the remaining return values
    out->outPublic.publicArea = newObject->publicArea;

    return TPM_RC_SUCCESS;
}

#endif  // CC_Create
```

7.103 /tpm/src/command/Object/CreateLoaded.c

```
#include "Tpm.h"
#include "CreateLoaded_fp.h"

#if CC_CreateLoaded  // Conditional expansion of this file

/*(See part 3 of specification)
 * Create and load any type of key, including a temporary key.
 * The input template is a marshaled public area rather than an unmarshaled one as
```

```
 * used in Create and CreatePrimary. This is so that the label and context that
 * could be in the template can be processed without changing the formats for the
 * calls to Create and CreatePrimary.
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'sensitiveDataOrigin' is CLEAR when 'sensitive.data'
//                              is an Empty Buffer;
//                              'fixedTPM', 'fixedParent', or 'encryptedDuplication'
//                              attributes are inconsistent between themselves or with
//                              those of the parent object;
//                              inconsistent 'restricted', 'decrypt' and 'sign'
//                              attributes;
//                              attempt to inject sensitive data for an asymmetric
//                              key;
//                              attempt to create a symmetric cipher key that is not
//                              a decryption key
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM failed
//                              to derive the Firmware SVN Secret for the requested
//                              SVN.
//      TPM_RC_KDF              incorrect KDF specified for decrypting keyed hash
//                              object
//      TPM_RC_KEY              the value of a provided symmetric key is not allowed
//      TPM_RC_OBJECT_MEMORY    there is no free slot for the object
//      TPM_RC_SCHEME           inconsistent attributes 'decrypt', 'sign',
//                              'restricted' and key's scheme ID; or hash algorithm is
//                              inconsistent with the scheme ID for keyed hash object
//      TPM_RC_SIZE             size of public authorization policy or sensitive
//                              authorization value does not match digest size of the
//                              name algorithm sensitive data size for the keyed hash
//                              object is larger than is allowed for the scheme
//      TPM_RC_SYMMETRIC        a storage key with no symmetric algorithm specified;
//                              or non-storage key with symmetric algorithm different
//                              from TPM_ALG_NULL
//      TPM_RC_TYPE             cannot create the object of the indicated type
//                              (usually only occurs if trying to derive an RSA key).
TPM_RC
TPM2_CreateLoaded(CreateLoaded_In*  in,  // IN: input parameter list
                  CreateLoaded_Out* out  // OUT: output parameter list
)
{
    TPM_RC        result = TPM_RC_SUCCESS;
    OBJECT*       parent = HandleToObject(in->parentHandle);
    OBJECT*       newObject;
    BOOL          derivation;
    TPMT_PUBLIC* publicArea;
    RAND_STATE    randState;
    RAND_STATE*   rand = &randState;
    TPMS_DERIVE   labelContext;

    // Input Validation

    // How the public area is unmarshaled is determined by the parent, so
    // see if parent is a derivation parent
    derivation = (parent != NULL && parent->attributes.derivation);

    // If the parent is an object, then make sure that it is either a parent or
    // derivation parent
    if(parent != NULL && !parent->attributes.isParent && !derivation)
        return TPM_RCS_TYPE + RC_CreateLoaded_parentHandle;

    // Get a spot in which to create the newObject
    newObject = FindEmptyObjectSlot(&out->objectHandle);
```

```
    if(newObject == NULL)
        return TPM_RC_OBJECT_MEMORY;

    // Do this to save typing
    publicArea = &newObject->publicArea;

    // Unmarshal the template into the object space. TPM2_Create() and
    // TPM2_CreatePrimary() have the publicArea unmarshaled by CommandDispatcher.
    // This command is different because of an unfortunate property of the
    // unique field of an ECC key. It is a structure rather than a single TPM2B. If
    // if had been a TPM2B, then the label and context could be within a TPM2B and
    // unmarshaled like other public areas. Since it is not, this command needs its
    // on template that is a TPM2B that is unmarshaled as a BYTE array with a
    // its own unmarshal function.
    result = UnmarshalToPublic(publicArea, &in->inPublic, derivation, &labelContext);
    if(result != TPM_RC_SUCCESS)
        return result + RC_CreateLoaded_inPublic;

    // Validate that the authorization size is appropriate
    if(!AdjustAuthSize(&in->inSensitive.sensitive.userAuth, publicArea->nameAlg))
        return TPM_RCS_SIZE + RC_CreateLoaded_inSensitive;

    // Command output
    if(derivation)
    {
        TPMT_KEYEDHASH_SCHEME* scheme;
        scheme = &parent->publicArea.parameters.keyedHashDetail.scheme;

        // SP800-108 is the only KDF supported by this implementation and there is
        // no default hash algorithm.
        pAssert(scheme->details.xor.hashAlg != TPM_ALG_NULL
                && scheme->details.xor.kdf == TPM_ALG_KDF1_SP800_108);
        // Don't derive RSA keys
        if(publicArea->type == TPM_ALG_RSA)
            return TPM_RCS_TYPE + RC_CreateLoaded_inPublic;
        // sensitiveDataOrigin has to be CLEAR in a derived object. Since this
        // is specific to a derived object, it is checked here.
        if(IS_ATTRIBUTE(
               publicArea->objectAttributes, TPMA_OBJECT, sensitiveDataOrigin))
            return TPM_RCS_ATTRIBUTES;
        // Check the rest of the attributes
        result = PublicAttributesValidation(parent, 0, publicArea);
        if(result != TPM_RC_SUCCESS)
            return RcSafeAddToResult(result, RC_CreateLoaded_inPublic);
        // Process the template and sensitive areas to get the actual 'label' and
        // 'context' values to be used for this derivation.
        result = SetLabelAndContext(&labelContext, &in->inSensitive.sensitive.data);
        if(result != TPM_RC_SUCCESS)
            return result;
        // Set up the KDF for object generation
        DRBG_InstantiateSeededKdf((KDF_STATE*)rand,
                                  scheme->details.xor.hashAlg,
                                  scheme->details.xor.kdf,
                                  &parent->sensitive.sensitive.bits.b,
                                  &labelContext.label.b,
                                  &labelContext.context.b,
                                  TPM_MAX_DERIVATION_BITS);
        // Clear the sensitive size so that the creation functions will not try
        // to use this value.
        in->inSensitive.sensitive.data.t.size = 0;
    }
    else
    {
        // Check attributes in input public area. CreateChecks() checks the things
        // that are unique to creation and then validates the attributes and values
        // that are common to create and load.
```

```
        result = CreateChecks(parent,
                              (parent == NULL) ? in->parentHandle : 0,
                              publicArea,
                              in->inSensitive.sensitive.data.t.size);

        if(result != TPM_RC_SUCCESS)
            return RcSafeAddToResult(result, RC_CreateLoaded_inPublic);
        // Creating a primary object
        if(parent == NULL)
        {
            TPM2B_NAME name;
            TPM2B_SEED primary_seed;

            newObject->attributes.primary = SET;
            if(HierarchyNormalizeHandle(in->parentHandle) == TPM_RH_ENDORSEMENT)
                newObject->attributes.epsHierarchy = SET;

            result = HierarchyGetPrimarySeed(in->parentHandle, &primary_seed);
            if(result != TPM_RC_SUCCESS)
                return result;

            // If so, use the primary seed and the digest of the template
            // to seed the DRBG
            result = DRBG_InstantiateSeeded(
                (DRBG_STATE*)rand,
                &primary_seed.b,
                PRIMARY_OBJECT_CREATION,
                (TPM2B*)PublicMarshalAndComputeName(publicArea, &name),
                &in->inSensitive.sensitive.data.b);
            MemorySet(primary_seed.b.buffer, 0, primary_seed.b.size);

            if(result != TPM_RC_SUCCESS)
                return result;
        }
        else
        {
            // This is an ordinary object so use the normal random number generator
            rand = NULL;
        }
    }
    // Internal data update
    // Create the object
    result = CryptCreateObject(newObject, &in->inSensitive.sensitive, rand);
    DRBG_Uninstantiate((DRBG_STATE*)rand);
    if(result != TPM_RC_SUCCESS)
        return result;
    // if this is not a Primary key and not a derived key, then return the sensitive
    // area
    if(parent != NULL && !derivation)
        // Prepare output private data from sensitive
        SensitiveToPrivate(&newObject->sensitive,
                           &newObject->name,
                           parent,
                           newObject->publicArea.nameAlg,
                           &out->outPrivate);
    else
        out->outPrivate.t.size = 0;
    // Set the remaining return values
    out->outPublic.publicArea = newObject->publicArea;
    out->name                 = newObject->name;
    // Set the remaining attributes for a loaded object
    ObjectSetLoadedAttributes(newObject, in->parentHandle);

    return result;
}
```

```
#endif  // CC_CreateLoaded
```

7.104 /tpm/src/command/Object/Load.c

```
#include "Tpm.h"
#include "Load_fp.h"

#if CC_Load  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// Load an ordinary or temporary object
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'inPulblic' attributes are not allowed with selected
//                              parent
//      TPM_RC_BINDING          'inPrivate' and 'inPublic' are not
//                              cryptographically bound
//      TPM_RC_HASH             incorrect hash selection for signing key or
//                              the 'nameAlg' for 'inPublic' is not valid
//      TPM_RC_INTEGRITY        HMAC on 'inPrivate' was not valid
//      TPM_RC_KDF              KDF selection not allowed
//      TPM_RC_KEY              the size of the object's 'unique' field is not
//                              consistent with the indicated size in the object's
//                              parameters
//      TPM_RC_OBJECT_MEMORY    no available object slot
//      TPM_RC_SCHEME           the signing scheme is not valid for the key
//      TPM_RC_SENSITIVE        the 'inPrivate' did not unmarshal correctly
//      TPM_RC_SIZE             'inPrivate' missing, or 'authPolicy' size for
//                              'inPublic' or is not valid
//      TPM_RC_SYMMETRIC        symmetric algorithm not provided when required
//      TPM_RC_TYPE             'parentHandle' is not a storage key, or the object
//                              to load is a storage key but its parameters do not
//                              match the parameters of the parent.
//      TPM_RC_VALUE            decryption failure
TPM_RC
TPM2_Load(Load_In*  in,  // IN: input parameter list
          Load_Out* out  // OUT: output parameter list
)
{
    TPM_RC          result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE sensitive;
    OBJECT*         parentObject;
    OBJECT*         newObject;

    // Input Validation
    // Don't get invested in loading if there is no place to put it.
    newObject = FindEmptyObjectSlot(&out->objectHandle);
    if(newObject == NULL)
        return TPM_RC_OBJECT_MEMORY;

    if(in->inPrivate.t.size == 0)
        return TPM_RCS_SIZE + RC_Load_inPrivate;

    parentObject = HandleToObject(in->parentHandle);
    pAssert(parentObject != NULL);
    // Is the object that is being used as the parent actually a parent.
    if(!ObjectIsParent(parentObject))
        return TPM_RCS_TYPE + RC_Load_parentHandle;

    // Compute the name of object. If there isn't one, it is because the nameAlg is
    // not valid.
    PublicMarshalAndComputeName(&in->inPublic.publicArea, &out->name);
    if(out->name.t.size == 0)
```

```
        return TPM_RCS_HASH + RC_Load_inPublic;

    // Retrieve sensitive data.
    result = PrivateToSensitive(&in->inPrivate.b,
                                &out->name.b,
                                parentObject,
                                in->inPublic.publicArea.nameAlg,
                                &sensitive);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_Load_inPrivate);

    // Internal Data Update
    // Load and validate object
    result = ObjectLoad(newObject,
                        parentObject,
                        &in->inPublic.publicArea,
                        &sensitive,
                        RC_Load_inPublic,
                        RC_Load_inPrivate,
                        &out->name);
    if(result == TPM_RC_SUCCESS)
    {
        // Set the common OBJECT attributes for a loaded object.
        ObjectSetLoadedAttributes(newObject, in->parentHandle);
    }
    return result;
}

#endif  // CC_Load
```

7.105 /tpm/src/command/Object/LoadExternal.c

```
#include "Tpm.h"
#include "LoadExternal_fp.h"

#if CC_LoadExternal  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// to load an object that is not a Protected Object into the public portion
// of an object into the TPM. The command allows loading of a public area or
// both a public and sensitive area
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'fixedParent', 'fixedTPM', and 'restricted' must
//                              be CLEAR if sensitive portion of an object is loaded
//      TPM_RC_BINDING          the 'inPublic' and 'inPrivate' structures are not
//                              cryptographically bound
//      TPM_RC_HASH             incorrect hash selection for signing key
//      TPM_RC_HIERARCHY        'hierarchy' is turned off, or only NULL hierarchy
//                              is allowed when loading public and private parts
//                              of an object
//      TPM_RC_KDF              incorrect KDF selection for decrypting
//                              keyedHash object
//      TPM_RC_KEY              the size of the object's 'unique' field is not
//                              consistent with the indicated size in the object's
//                              parameters
//      TPM_RC_OBJECT_MEMORY    if there is no free slot for an object
//      TPM_RC_ECC_POINT        for a public-only ECC key, the ECC point is not
//                              on the curve
//      TPM_RC_SCHEME           the signing scheme is not valid for the key
//      TPM_RC_SIZE             'authPolicy' is not zero and is not the size of a
//                              digest produced by the object's 'nameAlg'
//                              TPM_RH_NULL hierarchy
```

```
//      TPM_RC_SYMMETRIC        symmetric algorithm not provided when required
//      TPM_RC_TYPE             'inPublic' and 'inPrivate' are not the same type
TPM_RC
TPM2_LoadExternal(LoadExternal_In*  in,  // IN: input parameter list
                  LoadExternal_Out* out  // OUT: output parameter list
)
{
    TPM_RC          result;
    OBJECT*         object;
    TPMT_SENSITIVE* sensitive = NULL;

    // Input Validation
    // Don't get invested in loading if there is no place to put it.
    object = FindEmptyObjectSlot(&out->objectHandle);
    if(object == NULL)
        return TPM_RC_OBJECT_MEMORY;

    // If the hierarchy to be associated with this object is turned off, the object
    // cannot be loaded.
    if(!HierarchyIsEnabled(in->hierarchy))
        return TPM_RCS_HIERARCHY + RC_LoadExternal_hierarchy;

    // For loading an object with both public and sensitive
    if(in->inPrivate.size != 0)
    {
        // An external object with a sensitive area can only be loaded in the
        // NULL hierarchy
        if(in->hierarchy != TPM_RH_NULL)
            return TPM_RCS_HIERARCHY + RC_LoadExternal_hierarchy;
        // An external object with a sensitive area must have fixedTPM == CLEAR
        // fixedParent == CLEAR so that it does not appear to be a key created by
        // this TPM.
        if(IS_ATTRIBUTE(
               in->inPublic.publicArea.objectAttributes, TPMA_OBJECT, fixedTPM)
           || IS_ATTRIBUTE(
               in->inPublic.publicArea.objectAttributes, TPMA_OBJECT, fixedParent)
           || IS_ATTRIBUTE(
               in->inPublic.publicArea.objectAttributes, TPMA_OBJECT, restricted))
            return TPM_RCS_ATTRIBUTES + RC_LoadExternal_inPublic;

        // Have sensitive point to something other than NULL so that object
        // initialization will load the sensitive part too
        sensitive = &in->inPrivate.sensitiveArea;
    }

    // Need the name to initialize the object structure
    PublicMarshalAndComputeName(&in->inPublic.publicArea, &out->name);

    // Load and validate key
    result = ObjectLoad(object,
                        NULL,
                        &in->inPublic.publicArea,
                        sensitive,
                        RC_LoadExternal_inPublic,
                        RC_LoadExternal_inPrivate,
                        &out->name);
    if(result == TPM_RC_SUCCESS)
    {
        object->attributes.external = SET;
        // Set the common OBJECT attributes for a loaded object.
        ObjectSetLoadedAttributes(object, in->hierarchy);
    }
    return result;
}

#endif  // CC_LoadExternal
```

7.106 /tpm/src/command/Object/MakeCredential.c

```
#include "Tpm.h"
#include "MakeCredential_fp.h"

#if CC_MakeCredential  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// Make Credential with an object
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY              'handle' referenced an ECC key that has a unique
//                              field that is not a point on the curve of the key
//      TPM_RC_SIZE             'credential' is larger than the digest size of
//                              Name algorithm of 'handle'
//      TPM_RC_TYPE             'handle' does not reference an asymmetric
//                              decryption key
TPM_RC
TPM2_MakeCredential(MakeCredential_In*  in,  // IN: input parameter list
                    MakeCredential_Out* out  // OUT: output parameter list
)
{
    TPM_RC     result = TPM_RC_SUCCESS;

    OBJECT*    object;
    TPM2B_DATA data;

    // Input Validation

    // Get object pointer
    object = HandleToObject(in->handle);

    // input key must be an asymmetric, restricted decryption key
    // NOTE: Needs to be restricted to have a symmetric value.
    if(!CryptIsAsymAlgorithm(object->publicArea.type)
       || !IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, decrypt)
       || !IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, restricted))
        return TPM_RCS_TYPE + RC_MakeCredential_handle;

    // The credential information may not be larger than the digest size used for
    // the Name of the key associated with handle.
    if(in->credential.t.size > CryptHashGetDigestSize(object->publicArea.nameAlg))
        return TPM_RCS_SIZE + RC_MakeCredential_credential;

    // Command Output

    // Make encrypt key and its associated secret structure.
    out->secret.t.size = sizeof(out->secret.t.secret);
    result = CryptSecretEncrypt(object, IDENTITY_STRING, &data, &out->secret);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Prepare output credential data from secret
    SecretToCredential(
        &in->credential, &in->objectName.b, &data.b, object, &out->credentialBlob);

    return TPM_RC_SUCCESS;
}

#endif  // CC_MakeCredential
```

7.107 /tpm/src/command/Object/ObjectChangeAuth.c

```
#include "Tpm.h"
#include "ObjectChangeAuth_fp.h"

#if CC_ObjectChangeAuth  // Conditional expansion of this file

#  include "Object_spt_fp.h"

/*(See part 3 specification)
// Create an object
*/
//  Return Type: TPM_RC
//      TPM_RC_SIZE             'newAuth' is larger than the size of the digest
//                              of the Name algorithm of 'objectHandle'
//      TPM_RC_TYPE             the key referenced by 'parentHandle' is not the
//                              parent of the object referenced by 'objectHandle';
//                              or 'objectHandle' is a sequence object.
TPM_RC
TPM2_ObjectChangeAuth(ObjectChangeAuth_In*  in,  // IN: input parameter list
                      ObjectChangeAuth_Out* out  // OUT: output parameter list
)
{
    TPMT_SENSITIVE sensitive;

    OBJECT*        object = HandleToObject(in->objectHandle);
    TPM2B_NAME     QNCompare;

    // Input Validation

    // Can not change authorization on sequence object
    if(ObjectIsSequence(object))
        return TPM_RCS_TYPE + RC_ObjectChangeAuth_objectHandle;

    // Make sure that the authorization value is consistent with the nameAlg
    if(!AdjustAuthSize(&in->newAuth, object->publicArea.nameAlg))
        return TPM_RCS_SIZE + RC_ObjectChangeAuth_newAuth;

    // Parent handle should be the parent of object handle.  In this
    // implementation we verify this by checking the QN of object.  Other
    // implementation may choose different method to verify this attribute.
    ComputeQualifiedName(
        in->parentHandle, object->publicArea.nameAlg, &object->name, &QNCompare);
    if(!MemoryEqual2B(&object->qualifiedName.b, &QNCompare.b))
        return TPM_RCS_TYPE + RC_ObjectChangeAuth_parentHandle;

    // Command Output
    // Prepare the sensitive area with the new authorization value
    sensitive           = object->sensitive;
    sensitive.authValue = in->newAuth;

    // Protect the sensitive area
    SensitiveToPrivate(&sensitive,
                       &object->name,
                       HandleToObject(in->parentHandle),
                       object->publicArea.nameAlg,
                       &out->outPrivate);
    return TPM_RC_SUCCESS;
}

#endif  // CC_ObjectChangeAuth
```

7.108 /tpm/src/command/Object/Object_spt.c

```
//** Includes
```

```
#include "Tpm.h"
#include "Object_spt_fp.h"
#include "Marshal.h"

//** Local Functions

//*** GetIV2BSize()
// Get the size of TPM2B_IV in canonical form that will be append to the start of
// the sensitive data.  It includes both size of size field and size of iv data
static UINT16 GetIV2BSize(OBJECT* protector  // IN: the protector handle
)
{
    TPM_ALG_ID symAlg;
    UINT16     keyBits;

    // Determine the symmetric algorithm and size of key
    if(protector == NULL)
    {
        // Use the context encryption algorithm and key size
        symAlg  = CONTEXT_ENCRYPT_ALG;
        keyBits = CONTEXT_ENCRYPT_KEY_BITS;
    }
    else
    {
        symAlg  = protector->publicArea.parameters.asymDetail.symmetric.algorithm;
        keyBits = protector->publicArea.parameters.asymDetail.symmetric.keyBits.sym;
    }

    // The IV size is a UINT16 size field plus the block size of the symmetric
    // algorithm
    return sizeof(UINT16) + CryptGetSymmetricBlockSize(symAlg, keyBits);
}

//*** ComputeProtectionKeyParms()
// This function retrieves the symmetric protection key parameters for
// the sensitive data
// The parameters retrieved from this function include encryption algorithm,
// key size in bit, and a TPM2B_SYM_KEY containing the key material as well as
// the key size in bytes
// This function is used for any action that requires encrypting or decrypting of
// the sensitive area of an object or a credential blob
//
/*(See part 1 specification)
    KDF for generating the protection key material:
    KDFa(hashAlg, seed, "STORAGE", Name, NULL , bits)
where
    hashAlg     for a Primary Object, an algorithm chosen by the TPM vendor
                for derivations from Primary Seeds. For all other objects,
                the nameAlg of the object's parent.
    seed        for a Primary Object in the Platform Hierarchy, the PPS.
                For Primary Objects in either Storage or Endorsement Hierarchy,
                the SPS. For Temporary Objects, the context encryption seed.
                For all other objects, the symmetric seed value in the
                sensitive area of the object's parent.
    STORAGE     label to differentiate use of KDFa() (see 4.7)
    Name        the Name of the object being encrypted
    bits        the number of bits required for a  symmetric key and IV
*/
//  Return Type: void
static void ComputeProtectionKeyParms(
    OBJECT*    protector,    // IN: the protector object
    TPM_ALG_ID hashAlg,      // IN: hash algorithm for KDFa
    TPM2B*     name,         // IN: name of the object
    TPM2B*     seedIn,       // IN: optional seed for duplication blob.
                             //     For non duplication blob, this
                             //     parameter should be NULL
```

```
    TPM_ALG_ID*     symAlg,    // OUT: the symmetric algorithm
    UINT16*         keyBits,   // OUT: the symmetric key size in bits
    TPM2B_SYM_KEY* symKey     // OUT: the symmetric key
)
{
    const TPM2B* seed = seedIn;

    // Determine the algorithms for the KDF and the encryption/decryption
    // For TPM_RH_NULL, using context settings
    if(protector == NULL)
    {
        // Use the context encryption algorithm and key size
        *symAlg        = CONTEXT_ENCRYPT_ALG;
        symKey->t.size = CONTEXT_ENCRYPT_KEY_BYTES;
        *keyBits       = CONTEXT_ENCRYPT_KEY_BITS;
    }
    else
    {
        TPMT_SYM_DEF_OBJECT* symDef;
        symDef         = &protector->publicArea.parameters.asymDetail.symmetric;
        *symAlg        = symDef->algorithm;
        *keyBits       = symDef->keyBits.sym;
        symKey->t.size = (*keyBits + 7) / 8;
    }
    // Get seed for KDF
    if(seed == NULL)
        seed = GetSeedForKDF(protector);
    // KDFa to generate symmetric key and IV value
    CryptKDFa(hashAlg,
              seed,
              STORAGE_KEY,
              name,
              NULL,
              symKey->t.size * 8,
              symKey->t.buffer,
              NULL,
              FALSE);
    return;
}

//*** ComputeOuterIntegrity()
// The sensitive area parameter is a buffer that holds a space for
// the integrity value and the marshaled sensitive area. The caller should
// skip over the area set aside for the integrity value
// and compute the hash of the remainder of the object.
// The size field of sensitive is in unmarshaled form and the
// sensitive area contents is an array of bytes.
/*(See part 1 specification)
    KDFa(hashAlg, seed, "INTEGRITY", NULL, NULL , bits)   (38)
where
    hashAlg     for a Primary Object, the nameAlg of the object. For all other
                objects the nameAlg of the object's parent.
    seed        for a Primary Object in the Platform Hierarchy, the PPS. For
                Primary Objects in either Storage or Endorsement Hierarchy,
                the SPS. For a Temporary Object, the context encryption key.
                For all other objects, the symmetric seed value in the sensitive
                area of the object's parent.
    "INTEGRITY" a value used to differentiate the uses of the KDF.
    bits        the number of bits in the digest produced by hashAlg.
Key is then used in the integrity computation.
    HMACnameAlg(HMACkey, encSensitive || Name )
where
    HMACnameAlg()   the HMAC function using nameAlg of the object's parent
    HMACkey         value derived from the parent symmetric protection value
    encSensitive    symmetrically encrypted sensitive area
    Name            the Name of the object being protected
```

```
*/
//  Return Type: void
static void ComputeOuterIntegrity(
    TPM2B*  name,                   // IN: the name of the object
    OBJECT* protector,              // IN: the object that
                                    //     provides protection. For an object,
                                    //     it is a parent. For a credential, it
                                    //     is the encrypt object. For
                                    //     a Temporary Object, it is NULL
    TPMI_ALG_HASH hashAlg,          // IN: algorithm to use for integrity
    TPM2B*        seedIn,           // IN: an external seed may be provided for
                                    //     duplication blob. For non duplication
                                    //     blob, this parameter should be NULL
    UINT32        sensitiveSize,    // IN: size of the marshaled sensitive data
    BYTE*         sensitiveData,    // IN: sensitive area
    TPM2B_DIGEST* integrity         // OUT: integrity
)
{
    HMAC_STATE   hmacState;
    TPM2B_DIGEST hmacKey;
    const TPM2B* seed = seedIn;
    //
    // Get seed for KDF
    if(seed == NULL)
        seed = GetSeedForKDF(protector);
    // Determine the HMAC key bits
    hmacKey.t.size = CryptHashGetDigestSize(hashAlg);

    // KDFa to generate HMAC key
    CryptKDFa(hashAlg,
              seed,
              INTEGRITY_KEY,
              NULL,
              NULL,
              hmacKey.t.size * 8,
              hmacKey.t.buffer,
              NULL,
              FALSE);
    // Start HMAC and get the size of the digest which will become the integrity
    integrity->t.size = CryptHmacStart2B(&hmacState, hashAlg, &hmacKey.b);

    // Adding the marshaled sensitive area to the integrity value
    CryptDigestUpdate(&hmacState.hashState, sensitiveSize, sensitiveData);

    // Adding name
    CryptDigestUpdate2B(&hmacState.hashState, name);

    // Compute HMAC
    CryptHmacEnd2B(&hmacState, &integrity->b);

    return;
}

//*** ComputeInnerIntegrity()
// This function computes the integrity of an inner wrap
static void ComputeInnerIntegrity(
    TPM_ALG_ID    hashAlg,          // IN: hash algorithm for inner wrap
    TPM2B*        name,             // IN: the name of the object
    UINT16        dataSize,         // IN: the size of sensitive data
    BYTE*         sensitiveData,    // IN: sensitive data
    TPM2B_DIGEST* integrity         // OUT: inner integrity
)
{
    HASH_STATE hashState;
    //
    // Start hash and get the size of the digest which will become the integrity
```

```
    integrity->t.size = CryptHashStart(&hashState, hashAlg);

    // Adding the marshaled sensitive area to the integrity value
    CryptDigestUpdate(&hashState, dataSize, sensitiveData);

    // Adding name
    CryptDigestUpdate2B(&hashState, name);

    // Compute hash
    CryptHashEnd2B(&hashState, &integrity->b);

    return;
}

//*** ProduceInnerIntegrity()
// This function produces an inner integrity for regular private, credential or
// duplication blob
// It requires the sensitive data being marshaled to the innerBuffer, with the
// leading bytes reserved for integrity hash.  It assume the sensitive data
// starts at address (innerBuffer + integrity size).
// This function integrity at the beginning of the inner buffer
// It returns the total size of buffer with the inner wrap
static UINT16 ProduceInnerIntegrity(
    TPM2B*     name,      // IN: the name of the object
    TPM_ALG_ID hashAlg,   // IN: hash algorithm for inner wrap
    UINT16     dataSize,  // IN: the size of sensitive data, excluding the
                          //     leading integrity buffer size
    BYTE* innerBuffer     // IN/OUT: inner buffer with sensitive data in
                          //     it.  At input, the leading bytes of this
                          //     buffer is reserved for integrity
)
{
    BYTE*        sensitiveData;  // pointer to the sensitive data
    TPM2B_DIGEST integrity;
    UINT16       integritySize;
    BYTE*        buffer;  // Auxiliary buffer pointer
                          //
    // sensitiveData points to the beginning of sensitive data in innerBuffer
    integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);
    sensitiveData = innerBuffer + integritySize;

    ComputeInnerIntegrity(hashAlg, name, dataSize, sensitiveData, &integrity);

    // Add integrity at the beginning of inner buffer
    buffer = innerBuffer;
    TPM2B_DIGEST_Marshal(&integrity, &buffer, NULL);

    return dataSize + integritySize;
}

//*** CheckInnerIntegrity()
// This function check integrity of inner blob
//  Return Type: TPM_RC
//      TPM_RC_INTEGRITY        if the outer blob integrity is bad
//      unmarshal errors        unmarshal errors while unmarshaling integrity
static TPM_RC CheckInnerIntegrity(
    TPM2B*     name,      // IN: the name of the object
    TPM_ALG_ID hashAlg,   // IN: hash algorithm for inner wrap
    UINT16     dataSize,  // IN: the size of sensitive data, including the
                          //     leading integrity buffer size
    BYTE* innerBuffer     // IN/OUT: inner buffer with sensitive data in
                          //     it
)
{
    TPM_RC       result;
    TPM2B_DIGEST integrity;
```

```
    TPM2B_DIGEST integrityToCompare;
    BYTE*        buffer;  // Auxiliary buffer pointer
    INT32        size;
    //
    // Unmarshal integrity
    buffer = innerBuffer;
    size   = (INT32)dataSize;
    result = TPM2B_DIGEST_Unmarshal(&integrity, &buffer, &size);
    if(result == TPM_RC_SUCCESS)
    {
        // Compute integrity to compare
        ComputeInnerIntegrity(
            hashAlg, name, (UINT16)size, buffer, &integrityToCompare);
        // Compare outer blob integrity
        if(!MemoryEqual2B(&integrity.b, &integrityToCompare.b))
            result = TPM_RC_INTEGRITY;
    }
    return result;
}

//** Public Functions

//*** AdjustAuthSize()
// This function will validate that the input authValue is no larger than the
// digestSize for the nameAlg. It will then pad with zeros to the size of the
// digest.
BOOL AdjustAuthSize(TPM2B_AUTH*   auth,    // IN/OUT: value to adjust
                    TPMI_ALG_HASH nameAlg  // IN:
)
{
    UINT16 digestSize;
    //
    // If there is no nameAlg, then this is a LoadExternal and the authVale can
    // be any size up to the maximum allowed by the implementation
    digestSize = (nameAlg == TPM_ALG_NULL) ? sizeof(TPMU_HA)
                                           : CryptHashGetDigestSize(nameAlg);
    if(digestSize < MemoryRemoveTrailingZeros(auth))
        return FALSE;
    else if(digestSize > auth->t.size)
        MemoryPad2B(&auth->b, digestSize);
    auth->t.size = digestSize;

    return TRUE;
}

//*** AreAttributesForParent()
// This function is called by create, load, and import functions.
//
// Note: The 'isParent' attribute is SET when an object is loaded and it has
// attributes that are suitable for a parent object.
//  Return Type: BOOL
//      TRUE(1)         properties are those of a parent
//      FALSE(0)        properties are not those of a parent
BOOL ObjectIsParent(OBJECT* parentObject  // IN: parent handle
)
{
    return parentObject->attributes.isParent;
}

//*** CreateChecks()
// Attribute checks that are unique to creation.
// If parentObject is not NULL, then this function checks the object's
// attributes as an Ordinary or Derived Object with the given parent.
// If parentObject is NULL, and primaryHandle is not 0, then this function
// checks the object's attributes as a Primary Object in the given hierarchy.
// If parentObject is NULL, and primaryHandle is 0, then this function checks
```

```
// the object's attributes as an External Object.
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   sensitiveDataOrigin is not consistent with the
//                          object type
//      other               returns from PublicAttributesValidation()
TPM_RC
CreateChecks(OBJECT*           parentObject,
             TPMI_RH_HIERARCHY primaryHierarchy,
             TPMT_PUBLIC*      publicArea,
             UINT16            sensitiveDataSize)
{
    TPMA_OBJECT attributes = publicArea->objectAttributes;
    TPM_RC      result     = TPM_RC_SUCCESS;
    //
    // If the caller indicates that they have provided the data, then make sure that
    // they have provided some data.
    if((!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
       && (sensitiveDataSize == 0))
        return TPM_RCS_ATTRIBUTES;
    // For an ordinary object, data can only be provided when sensitiveDataOrigin
    // is CLEAR
    if((parentObject != NULL)
       && (IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
       && (sensitiveDataSize != 0))
        return TPM_RCS_ATTRIBUTES;
    switch(publicArea->type)
    {
        case TPM_ALG_KEYEDHASH:
            // if this is a data object (sign == decrypt == CLEAR) then the
            // TPM cannot be the data source.
            if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
               && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt)
               && IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
                result = TPM_RC_ATTRIBUTES;
            // comment out the next line in order to prevent a fixedTPM derivation
            // parent
            //            break;
        case TPM_ALG_SYMCIPHER:
            // A restricted key symmetric key (SYMCIPHER and KEYEDHASH)
            // must have sensitiveDataOrigin SET unless it has fixedParent and
            // fixedTPM CLEAR.
            if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
                if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
                    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent)
                       || IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM))
                        result = TPM_RCS_ATTRIBUTES;
            break;
        default:  // Asymmetric keys cannot have the sensitive portion provided
            if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
                result = TPM_RCS_ATTRIBUTES;
            break;
    }
    if(TPM_RC_SUCCESS == result)
    {
        result =
            PublicAttributesValidation(parentObject, primaryHierarchy, publicArea);
    }
    return result;
}

//*** SchemeChecks
// This function is called by TPM2_LoadExternal() and PublicAttributesValidation().
// This function validates the schemes in the public area of an object.
//  Return Type: TPM_RC
//      TPM_RC_HASH         non-duplicable storage key and its parent have different
//                          name algorithm
```

```
//      TPM_RC_KDF          incorrect KDF specified for decrypting keyed hash object
//      TPM_RC_KEY          invalid key size values in an asymmetric key public area
//      TPM_RCS_SCHEME       inconsistent attributes 'decrypt', 'sign', 'restricted'
//                          and key's scheme ID; or hash algorithm is inconsistent
//                          with the scheme ID for keyed hash object
//      TPM_RC_SYMMETRIC    a storage key with no symmetric algorithm specified; or
//                          non-storage key with symmetric algorithm different from
//                          TPM_ALG_NULL
TPM_RC
SchemeChecks(OBJECT*      parentObject,  // IN: parent (null if primary seed)
             TPMT_PUBLIC* publicArea     // IN: public area of the object
)
{
    TPMT_SYM_DEF_OBJECT* symAlgs    = NULL;
    TPM_ALG_ID           scheme     = TPM_ALG_NULL;
    TPMA_OBJECT          attributes = publicArea->objectAttributes;
    TPMU_PUBLIC_PARMS*   parms      = &publicArea->parameters;
    //
    switch(publicArea->type)
    {
        case TPM_ALG_SYMCIPHER:
            symAlgs = &parms->symDetail.sym;
            // If this is a decrypt key, then only the block cipher modes (not
            // SMAC) are valid. TPM_ALG_NULL is OK too. If this is a 'sign' key,
            // then any mode that got through the unmarshaling is OK.
            if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt)
               && !CryptSymModeIsValid(symAlgs->mode.sym, TRUE))
                return TPM_RCS_SCHEME;
            break;
        case TPM_ALG_KEYEDHASH:
            scheme = parms->keyedHashDetail.scheme.scheme;
            // if both sign and decrypt
            if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
               == IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
            {
                // if both sign and decrypt are set or clear, then need
                // TPM_ALG_NULL as scheme
                if(scheme != TPM_ALG_NULL)
                    return TPM_RCS_SCHEME;
            }
            else if(
                IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign) && scheme != TPM_ALG_HMAC)
                return TPM_RCS_SCHEME;
            else if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
            {
                if(scheme != TPM_ALG_XOR)
                    return TPM_RCS_SCHEME;
                // If this is a derivation parent, then the KDF needs to be
                // SP800-108 for this implementation. This is the only derivation
                // supported by this implementation. Other implementations could
                // support additional schemes. There is no default.
                if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
                {
                    if(parms->keyedHashDetail.scheme.details.
                       xor.kdf != TPM_ALG_KDF1_SP800_108)
                        return TPM_RCS_SCHEME;
                    // Must select a digest.
                    if(CryptHashGetDigestSize(
                            parms->keyedHashDetail.scheme.details.xor.hashAlg)
                       == 0)
                        return TPM_RCS_HASH;
                }
            }
            break;
        default:  // handling for asymmetric
            scheme  = parms->asymDetail.scheme.scheme;
```

```
            symAlgs = &parms->asymDetail.symmetric;
            // if the key is both sign and decrypt, then the scheme must be
            // TPM_ALG_NULL because there is no way to specify both a sign and a
            // decrypt scheme in the key.
            if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
               == IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
            {
                // scheme must be TPM_ALG_NULL
                if(scheme != TPM_ALG_NULL)
                    return TPM_RCS_SCHEME;
            }
            else if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign))
            {
                // If this is a signing key, see if it has a signing scheme
                if(CryptIsAsymSignScheme(publicArea->type, scheme))
                {
                    // if proper signing scheme then it needs a proper hash
                    if(parms->asymDetail.scheme.details.anySig.hashAlg
                       == TPM_ALG_NULL)
                        return TPM_RCS_SCHEME;
                }
                else
                {
                    // signing key that does not have a proper signing scheme.
                    // This is OK if the key is not restricted and its scheme
                    // is TPM_ALG_NULL
                    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted)
                       || scheme != TPM_ALG_NULL)
                        return TPM_RCS_SCHEME;
                }
            }
            else if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
            {
                if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
                {
                    // for a restricted decryption key (a parent), scheme
                    // is required to be TPM_ALG_NULL
                    if(scheme != TPM_ALG_NULL)
                        return TPM_RCS_SCHEME;
                }
                else
                {
                    // For an unrestricted decryption key, the scheme has to
                    // be a valid scheme or TPM_ALG_NULL
                    if(scheme != TPM_ALG_NULL
                       && !CryptIsAsymDecryptScheme(publicArea->type, scheme))
                        return TPM_RCS_SCHEME;
                }
            }
            if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted)
               || !IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
            {
                // For an asymmetric key that is not a parent, the symmetric
                // algorithms must be TPM_ALG_NULL
                if(symAlgs->algorithm != TPM_ALG_NULL)
                    return TPM_RCS_SYMMETRIC;
            }
            // Special checks for an ECC key
#if ALG_ECC
            if(publicArea->type == TPM_ALG_ECC)
            {
                TPM_ECC_CURVE          curveID;
                const TPMT_ECC_SCHEME* curveScheme;

                curveID     = publicArea->parameters.eccDetail.curveID;
                curveScheme = CryptGetCurveSignScheme(curveID);
```

```
                // The curveId must be valid or the unmarshaling is busted.
                pAssert(curveScheme != NULL);

                // If the curveID requires a specific scheme, then the key must
                // select the same scheme
                if(curveScheme->scheme != TPM_ALG_NULL)
                {
                    TPMS_ECC_PARMS* ecc = &publicArea->parameters.eccDetail;
                    if(scheme != curveScheme->scheme)
                        return TPM_RCS_SCHEME;
                    // The scheme can allow any hash, or not...
                    if(curveScheme->details.anySig.hashAlg != TPM_ALG_NULL
                       && (ecc->scheme.details.anySig.hashAlg
                           != curveScheme->details.anySig.hashAlg))
                        return TPM_RCS_SCHEME;
                }
                // For now, the KDF must be TPM_ALG_NULL
                if(publicArea->parameters.eccDetail.kdf.scheme != TPM_ALG_NULL)
                    return TPM_RCS_KDF;
            }
#endif
            break;
    }
    // If this is a restricted decryption key with symmetric algorithms, then it
    // is an ordinary parent (not a derivation parent). It needs to specific
    // symmetric algorithms other than TPM_ALG_NULL
    if(symAlgs != NULL && IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted)
       && IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
    {
        if(symAlgs->algorithm == TPM_ALG_NULL)
            return TPM_RCS_SYMMETRIC;
#if 0  //??
// This next check is under investigation. Need to see if it will break Windows
// before it is enabled. If it does not, then it should be default because a
// the mode used with a parent is always CFB and Part 2 indicates as much.
        if(symAlgs->mode.sym != TPM_ALG_CFB)
            return TPM_RCS_MODE;
#endif
        // If this parent is not duplicable, then the symmetric algorithms
        // (encryption and hash) must match those of its parent
        if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent)
           && (parentObject != NULL))
        {
            if(publicArea->nameAlg != parentObject->publicArea.nameAlg)
                return TPM_RCS_HASH;
            if(!MemoryEqual(symAlgs,
                            &parentObject->publicArea.parameters,
                            sizeof(TPMT_SYM_DEF_OBJECT)))
                return TPM_RCS_SYMMETRIC;
        }
    }
    return TPM_RC_SUCCESS;
}

//*** PublicAttributesValidation()
// This function validates the values in the public area of an object.
// This function is used in the processing of TPM2_Create, TPM2_CreatePrimary,
// TPM2_CreateLoaded(), TPM2_Load(),  TPM2_Import(), and TPM2_LoadExternal().
// For TPM2_Import() this is only used if the new parent has fixedTPM SET. For
// TPM2_LoadExternal(), this is not used for a public-only key
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   'fixedTPM', 'fixedParent', or 'encryptedDuplication'
//                          attributes are inconsistent between themselves or with
//                          those of the parent object;
//                          inconsistent 'restricted', 'decrypt' and 'sign'
//                          attributes;
```

```
//                          attempt to inject sensitive data for an asymmetric key;
//                          attempt to create a symmetric cipher key that is not
//                          a decryption key
//      TPM_RC_HASH         nameAlg is TPM_ALG_NULL
//      TPM_RC_SIZE         'authPolicy' size does not match digest size of the name
//                          algorithm in 'publicArea'
//   other                  returns from SchemeChecks()
TPM_RC
PublicAttributesValidation(
    // IN: input parent object (if ordinary or derived object; NULL otherwise)
    OBJECT* parentObject,
    // IN: hierarchy (if primary object; 0 otherwise)
    TPMI_RH_HIERARCHY primaryHierarchy,
    // IN: public area of the object
    TPMT_PUBLIC* publicArea)
{
    TPMA_OBJECT attributes       = publicArea->objectAttributes;
    TPMA_OBJECT parentAttributes = TPMA_ZERO_INITIALIZER();

    if(parentObject != NULL)
        parentAttributes = parentObject->publicArea.objectAttributes;
    if(publicArea->nameAlg == TPM_ALG_NULL)
        return TPM_RCS_HASH;
    // If there is an authPolicy, it needs to be the size of the digest produced
    // by the nameAlg of the object
    if((publicArea->authPolicy.t.size != 0
        && (publicArea->authPolicy.t.size
            != CryptHashGetDigestSize(publicArea->nameAlg))))
        return TPM_RCS_SIZE;
    // If the parent is fixedTPM (including a Primary Object) the object must have
    // the same value for fixedTPM and fixedParent
    if(parentObject == NULL || IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, fixedTPM))
    {
        if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent)
           != IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM))
            return TPM_RCS_ATTRIBUTES;
    }
    else
    {
        // The parent is not fixedTPM so the object can't be fixedTPM
        if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM))
            return TPM_RCS_ATTRIBUTES;
    }
    // See if sign and decrypt are the same
    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
       == IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
    {
        // a restricted key cannot have both SET or both CLEAR
        if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
            return TPM_RC_ATTRIBUTES;
        // only a data object may have both sign and decrypt CLEAR
        // BTW, since we know that decrypt==sign, no need to check both
        if(publicArea->type != TPM_ALG_KEYEDHASH
           && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign))
            return TPM_RC_ATTRIBUTES;
    }
    // If the object can't be duplicated (directly or indirectly) then there
    // is no justification for having encryptedDuplication SET
    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM)
       && IS_ATTRIBUTE(attributes, TPMA_OBJECT, encryptedDuplication))
        return TPM_RCS_ATTRIBUTES;
    // If a parent object has fixedTPM CLEAR, the child must have the
    // same encryptedDuplication value as its parent.
    // Primary objects are considered to have a fixedTPM parent (the seeds).
    if(parentObject != NULL && !IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, fixedTPM))
    {
```

```
    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, encryptedDuplication)
       != IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, encryptedDuplication))
        return TPM_RCS_ATTRIBUTES;
}
// firmwareLimited/svnLimited can only be set if fixedTPM is also set.
if((IS_ATTRIBUTE(attributes, TPMA_OBJECT, firmwareLimited)
    || IS_ATTRIBUTE(attributes, TPMA_OBJECT, svnLimited))
   && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM))
{
    return TPM_RCS_ATTRIBUTES;
}

// firmwareLimited/svnLimited also impose requirements on the parent key or
// primary handle.
if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, firmwareLimited))
{
    if(parentObject != NULL)
    {
        // For an ordinary object, firmwareLimited can only be set if its
        // parent is also firmwareLimited.
        if(!IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, firmwareLimited))
            return TPM_RCS_ATTRIBUTES;
    }
    else if(primaryHierarchy != 0)
    {
        // For a primary object, firmwareLimited can only be set if its
        // hierarchy is a firmware-limited hierarchy.
        if(!HierarchyIsFirmwareLimited(primaryHierarchy))
            return TPM_RCS_ATTRIBUTES;
    }
    else
    {
        return TPM_RCS_ATTRIBUTES;
    }
}
if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, svnLimited))
{
    if(parentObject != NULL)
    {
        // For an ordinary object, svnLimited can only be set if its
        // parent is also svnLimited.
        if(!IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, svnLimited))
            return TPM_RCS_ATTRIBUTES;
    }
    else if(primaryHierarchy != 0)
    {
        // For a primary object, svnLimited can only be set if its
        // hierarchy is an svn-limited hierarchy.
        if(!HierarchyIsSvnLimited(primaryHierarchy))
            return TPM_RCS_ATTRIBUTES;
    }
    else
    {
        return TPM_RCS_ATTRIBUTES;
    }
}

// Special checks for derived objects
if((parentObject != NULL) && (parentObject->attributes.derivation == SET))
{
    // A derived object has the same settings for fixedTPM as its parent
    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM)
       != IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, fixedTPM))
        return TPM_RCS_ATTRIBUTES;
    // A derived object is required to be fixedParent
    if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent))
```

```
            return TPM_RCS_ATTRIBUTES;
    }
    return SchemeChecks(parentObject, publicArea);
}

//*** FillInCreationData()
// Fill in creation data for an object.
//  Return Type: void
void FillInCreationData(
    TPMI_DH_OBJECT       parentHandle,   // IN: handle of parent
    TPMI_ALG_HASH        nameHashAlg,    // IN: name hash algorithm
    TPML_PCR_SELECTION*  creationPCR,    // IN: PCR selection
    TPM2B_DATA*          outsideData,    // IN: outside data
    TPM2B_CREATION_DATA* outCreation,    // OUT: creation data for output
    TPM2B_DIGEST*        creationDigest  // OUT: creation digest
)
{
    BYTE       creationBuffer[sizeof(TPMS_CREATION_DATA)];
    BYTE*      buffer;
    HASH_STATE hashState;
    //
    // Fill in TPMS_CREATION_DATA in outCreation

    // Compute PCR digest
    PCRComputeCurrentDigest(
        nameHashAlg, creationPCR, &outCreation->creationData.pcrDigest);

    // Put back PCR selection list
    outCreation->creationData.pcrSelect = *creationPCR;

    // Get locality
    outCreation->creationData.locality = LocalityGetAttributes(_plat__LocalityGet());
    outCreation->creationData.parentNameAlg = TPM_ALG_NULL;

    // If the parent is either a primary seed or TPM_ALG_NULL, then  the Name
    // and QN of the parent are the parent's handle.
    if(HandleGetType(parentHandle) == TPM_HT_PERMANENT)
    {
        buffer = &outCreation->creationData.parentName.t.name[0];
        outCreation->creationData.parentName.t.size =
            TPM_HANDLE_Marshal(&parentHandle, &buffer, NULL);
        // For a primary or temporary object, the parent name (a handle) and the
        // parent's QN are the same
        outCreation->creationData.parentQualifiedName =
            outCreation->creationData.parentName;
    }
    else  // Regular object
    {
        OBJECT* parentObject = HandleToObject(parentHandle);
        //
        // Set name algorithm
        outCreation->creationData.parentNameAlg = parentObject->publicArea.nameAlg;

        // Copy parent name
        outCreation->creationData.parentName = parentObject->name;

        // Copy parent qualified name
        outCreation->creationData.parentQualifiedName = parentObject->qualifiedName;
    }
    // Copy outside information
    outCreation->creationData.outsideInfo = *outsideData;

    // Marshal creation data to canonical form
    buffer = creationBuffer;
    outCreation->size =
        TPMS_CREATION_DATA_Marshal(&outCreation->creationData, &buffer, NULL);
```

```
    // Compute hash for creation field in public template
    creationDigest->t.size = CryptHashStart(&hashState, nameHashAlg);
    CryptDigestUpdate(&hashState, outCreation->size, creationBuffer);
    CryptHashEnd2B(&hashState, &creationDigest->b);

    return;
}

//*** GetSeedForKDF()
// Get a seed for KDF.  The KDF for encryption and HMAC key use the same seed.
const TPM2B* GetSeedForKDF(OBJECT* protector  // IN: the protector handle
)
{
    // Get seed for encryption key.  Use input seed if provided.
    // Otherwise, using protector object's seedValue.  TPM_RH_NULL is the only
    // exception that we may not have a loaded object as protector.  In such a
    // case, use nullProof as seed.
    if(protector == NULL)
        return &gr.nullProof.b;
    else
        return &protector->sensitive.seedValue.b;
}

//*** ProduceOuterWrap()
// This function produce outer wrap for a buffer containing the sensitive data.
// It requires the sensitive data being marshaled to the outerBuffer, with the
// leading bytes reserved for integrity hash.  If iv is used, iv space should
// be reserved at the beginning of the buffer.  It assumes the sensitive data
// starts at address (outerBuffer + integrity size [+ iv size]).
// This function performs:
//  1. Add IV before sensitive area if required
//  2. encrypt sensitive data, if iv is required, encrypt by iv.  otherwise,
//     encrypted by a NULL iv
//  3. add HMAC integrity at the beginning of the buffer
// It returns the total size of blob with outer wrap
UINT16
ProduceOuterWrap(OBJECT* protector,    // IN: The handle of the object that provides
                                       //     protection.  For object, it is parent
                                       //     handle. For credential, it is the handle
                                       //     of encrypt object.
                 TPM2B*     name,      // IN: the name of the object
                 TPM_ALG_ID hashAlg,   // IN: hash algorithm for outer wrap
                 TPM2B*     seed,      // IN: an external seed may be provided for
                                       //     duplication blob. For non duplication
                                       //     blob, this parameter should be NULL
                 BOOL   useIV,         // IN: indicate if an IV is used
                 UINT16 dataSize,      // IN: the size of sensitive data, excluding the
                                       //     leading integrity buffer size or the
                                       //     optional iv size
                 BYTE* outerBuffer     // IN/OUT: outer buffer with sensitive data in
                                       //     it
)
{
    TPM_ALG_ID     symAlg;
    UINT16         keyBits;
    TPM2B_SYM_KEY symKey;
    TPM2B_IV       ivRNG;  // IV from RNG
    TPM2B_IV*      iv     = NULL;
    UINT16         ivSize = 0;     // size of iv area, including the size field
    BYTE*          sensitiveData;  // pointer to the sensitive data
    TPM2B_DIGEST   integrity;
    UINT16         integritySize;
    BYTE*          buffer;  // Auxiliary buffer pointer
                            //
    // Compute the beginning of sensitive data.  The outer integrity should
    // always exist if this function is called to make an outer wrap
```

```
    integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);
    sensitiveData = outerBuffer + integritySize;

    // If iv is used, adjust the pointer of sensitive data and add iv before it
    if(useIV)
    {
        ivSize = GetIV2BSize(protector);

        // Generate IV from RNG.  The iv data size should be the total IV area
        // size minus the size of size field
        ivRNG.t.size = ivSize - sizeof(UINT16);
        CryptRandomGenerate(ivRNG.t.size, ivRNG.t.buffer);

        // Marshal IV to buffer
        buffer = sensitiveData;
        TPM2B_IV_Marshal(&ivRNG, &buffer, NULL);

        // adjust sensitive data starting after IV area
        sensitiveData += ivSize;

        // Use iv for encryption
        iv = &ivRNG;
    }
    // Compute symmetric key parameters for outer buffer encryption
    ComputeProtectionKeyParms(
        protector, hashAlg, name, seed, &symAlg, &keyBits, &symKey);
    // Encrypt inner buffer in place
    CryptSymmetricEncrypt(sensitiveData,
                          symAlg,
                          keyBits,
                          symKey.t.buffer,
                          iv,
                          TPM_ALG_CFB,
                          dataSize,
                          sensitiveData);
    // Compute outer integrity.  Integrity computation includes the optional IV
    // area
    ComputeOuterIntegrity(name,
                          protector,
                          hashAlg,
                          seed,
                          dataSize + ivSize,
                          outerBuffer + integritySize,
                          &integrity);
    // Add integrity at the beginning of outer buffer
    buffer = outerBuffer;
    TPM2B_DIGEST_Marshal(&integrity, &buffer, NULL);

    // return the total size in outer wrap
    return dataSize + integritySize + ivSize;
}

//*** UnwrapOuter()
// This function remove the outer wrap of a blob containing sensitive data
// This function performs:
//  1. check integrity of outer blob
//  2. decrypt outer blob
//
//  Return Type: TPM_RC
//      TPM_RCS_INSUFFICIENT     error during sensitive data unmarshaling
//      TPM_RCS_INTEGRITY        sensitive data integrity is broken
//      TPM_RCS_SIZE             error during sensitive data unmarshaling
//      TPM_RCS_VALUE            IV size for CFB does not match the encryption
//                               algorithm block size
TPM_RC
UnwrapOuter(OBJECT* protector,   // IN: The object that provides
```

```
                                //     protection.  For object, it is parent
                                //     handle. For credential, it is the
                                //     encrypt object.
            TPM2B*     name,    // IN: the name of the object
            TPM_ALG_ID hashAlg, // IN: hash algorithm for outer wrap
            TPM2B*     seed,    // IN: an external seed may be provided for
                                //     duplication blob. For non duplication
                                //     blob, this parameter should be NULL.
            BOOL   useIV,       // IN: indicates if an IV is used
            UINT16 dataSize,    // IN: size of sensitive data in outerBuffer,
                                //     including the leading integrity buffer
                                //     size, and an optional iv area
            BYTE* outerBuffer   // IN/OUT: sensitive data
)
{
    TPM_RC        result;
    TPM_ALG_ID    symAlg = TPM_ALG_NULL;
    TPM2B_SYM_KEY symKey;
    UINT16        keyBits = 0;
    TPM2B_IV      ivIn;  // input IV retrieved from input buffer
    TPM2B_IV*     iv = NULL;
    BYTE*         sensitiveData;  // pointer to the sensitive data
    TPM2B_DIGEST  integrityToCompare;
    TPM2B_DIGEST  integrity;
    INT32         size;
    //
    // Unmarshal integrity
    sensitiveData = outerBuffer;
    size          = (INT32)dataSize;
    result        = TPM2B_DIGEST_Unmarshal(&integrity, &sensitiveData, &size);
    if(result == TPM_RC_SUCCESS)
    {
        // Compute integrity to compare
        ComputeOuterIntegrity(name,
                              protector,
                              hashAlg,
                              seed,
                              (UINT16)size,
                              sensitiveData,
                              &integrityToCompare);
        // Compare outer blob integrity
        if(!MemoryEqual2B(&integrity.b, &integrityToCompare.b))
            return TPM_RCS_INTEGRITY;
        // Get the symmetric algorithm parameters used for encryption
        ComputeProtectionKeyParms(
            protector, hashAlg, name, seed, &symAlg, &keyBits, &symKey);
        // Retrieve IV if it is used
        if(useIV)
        {
            result = TPM2B_IV_Unmarshal(&ivIn, &sensitiveData, &size);
            if(result == TPM_RC_SUCCESS)
            {
                // The input iv size for CFB must match the encryption algorithm
                // block size
                if(ivIn.t.size != CryptGetSymmetricBlockSize(symAlg, keyBits))
                    result = TPM_RC_VALUE;
                else
                    iv = &ivIn;
            }
        }
    }
    // If no errors, decrypt private in place. Since this function uses CFB,
    // CryptSymmetricDecrypt() will not return any errors. It may fail but it will
    // not return an error.
    if(result == TPM_RC_SUCCESS)
        CryptSymmetricDecrypt(sensitiveData,
```

```
                                symAlg,
                                keyBits,
                                symKey.t.buffer,
                                iv,
                                TPM_ALG_CFB,
                                (UINT16)size,
                                sensitiveData);
    return result;
}

//*** MarshalSensitive()
// This function is used to marshal a sensitive area. Among other things, it
// adjusts the size of the authValue to be no smaller than the digest of
// 'nameAlg'
// Returns the size of the marshaled area.
static UINT16 MarshalSensitive(
    OBJECT*         parent,     // IN: the object parent (optional)
    BYTE*           buffer,     // OUT: receiving buffer
    TPMT_SENSITIVE* sensitive,  // IN: the sensitive area to marshal
    TPMI_ALG_HASH   nameAlg     // IN:
)
{
    BYTE* sizeField = buffer;  // saved so that size can be
                               // marshaled after it is known
    UINT16 retVal;
    //
    // Pad the authValue if needed
    MemoryPad2B(&sensitive->authValue.b, CryptHashGetDigestSize(nameAlg));
    buffer += 2;

    // Marshal the structure
#if ALG_RSA
    // If the sensitive size is the special case for a prime in the type
    if((sensitive->sensitive.rsa.t.size & RSA_prime_flag) > 0)
    {
        UINT16 sizeSave = sensitive->sensitive.rsa.t.size;
        //
        // Turn off the flag that indicates that the sensitive->sensitive contains
        // the CRT form of the exponent.
        sensitive->sensitive.rsa.t.size &= ~(RSA_prime_flag);
        // If the parent isn't fixedTPM, then truncate the sensitive data to be
        // the size of the prime. Otherwise, leave it at the current size which
        // is the full CRT size.
        if(parent == NULL
           || !IS_ATTRIBUTE(
               parent->publicArea.objectAttributes, TPMA_OBJECT, fixedTPM))
            sensitive->sensitive.rsa.t.size /= 5;
        retVal = TPMT_SENSITIVE_Marshal(sensitive, &buffer, NULL);
        // Restore the flag and the size.
        sensitive->sensitive.rsa.t.size = sizeSave;
    }
    else
#endif
        retVal = TPMT_SENSITIVE_Marshal(sensitive, &buffer, NULL);

    // Marshal the size
    retVal = (UINT16)(retVal + UINT16_Marshal(&retVal, &sizeField, NULL));

    return retVal;
}

//*** SensitiveToPrivate()
// This function prepare the private blob for off the chip storage
// The operations in this function:
//  1. marshal TPM2B_SENSITIVE structure into the buffer of TPM2B_PRIVATE
//  2. apply encryption to the sensitive area.
```

```
//  3. apply outer integrity computation.
void SensitiveToPrivate(
    TPMT_SENSITIVE* sensitive,  // IN: sensitive structure
    TPM2B_NAME*     name,       // IN: the name of the object
    OBJECT*         parent,     // IN: The parent object
    TPM_ALG_ID      nameAlg,    // IN: hash algorithm in public area.  This
                                //     parameter is used when parentHandle is
                                //     NULL, in which case the object is
                                //     temporary.
    TPM2B_PRIVATE* outPrivate   // OUT: output private structure
)
{
    BYTE*         sensitiveData;  // pointer to the sensitive data
    UINT16        dataSize;       // data blob size
    TPMI_ALG_HASH hashAlg;        // hash algorithm for integrity
    UINT16        integritySize;
    UINT16        ivSize;
    //
    pAssert(name != NULL && name->t.size != 0);

    // Find the hash algorithm for integrity computation
    if(parent == NULL)
    {
        // For Temporary Object, using self name algorithm
        hashAlg = nameAlg;
    }
    else
    {
        // Otherwise, using parent's name algorithm
        hashAlg = parent->publicArea.nameAlg;
    }
    // Starting of sensitive data without wrappers
    sensitiveData = outPrivate->t.buffer;

    // Compute the integrity size
    integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);

    // Reserve space for integrity
    sensitiveData += integritySize;

    // Get iv size
    ivSize = GetIV2BSize(parent);

    // Reserve space for iv
    sensitiveData += ivSize;

    // Marshal the sensitive area including authValue size adjustments.
    dataSize = MarshalSensitive(parent, sensitiveData, sensitive, nameAlg);

    //Produce outer wrap, including encryption and HMAC
    outPrivate->t.size = ProduceOuterWrap(
        parent, &name->b, hashAlg, NULL, TRUE, dataSize, outPrivate->t.buffer);
    return;
}

//*** PrivateToSensitive()
// Unwrap an input private area; check the integrity; decrypt and retrieve data
// to a sensitive structure.
// The operations in this function:
//  1. check the integrity HMAC of the input private area
//  2. decrypt the private buffer
//  3. unmarshal TPMT_SENSITIVE structure into the buffer of TPMT_SENSITIVE
//
//  Return Type: TPM_RC
//      TPM_RCS_INTEGRITY       if the private area integrity is bad
//      TPM_RC_SENSITIVE        unmarshal errors while unmarshaling TPMS_ENCRYPT
```

```
//                              from input private
//      TPM_RCS_SIZE            error during sensitive data unmarshaling
//      TPM_RCS_VALUE           outer wrapper does not have an iV of the correct
//                              size
TPM_RC
PrivateToSensitive(TPM2B*     inPrivate,  // IN: input private structure
                   TPM2B*     name,       // IN: the name of the object
                   OBJECT*    parent,     // IN: parent object
                   TPM_ALG_ID nameAlg,    // IN: hash algorithm in public area.  It is
                   //     passed separately because we only pass
                   //     name, rather than the whole public area
                   //     of the object.  This parameter is used in
                   //     the following two cases: 1. primary
                   //     objects. 2. duplication blob with inner
                   //     wrap.  In other cases, this parameter
                   //     will be ignored
                   TPMT_SENSITIVE* sensitive  // OUT: sensitive structure
)
{
    TPM_RC        result;
    BYTE*         buffer;
    INT32         size;
    BYTE*         sensitiveData;  // pointer to the sensitive data
    UINT16        dataSize;
    UINT16        dataSizeInput;
    TPMI_ALG_HASH hashAlg;  // hash algorithm for integrity
    UINT16        integritySize;
    UINT16        ivSize;
    //
    // Make sure that name is provided
    pAssert(name != NULL && name->size != 0);

    // Find the hash algorithm for integrity computation
    // For Temporary Object (parent == NULL) use self name algorithm;
    // Otherwise, using parent's name algorithm
    hashAlg = (parent == NULL) ? nameAlg : parent->publicArea.nameAlg;

    // unwrap outer
    result = UnwrapOuter(
        parent, name, hashAlg, NULL, TRUE, inPrivate->size, inPrivate->buffer);
    if(result != TPM_RC_SUCCESS)
        return result;
    // Compute the inner integrity size.
    integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);

    // Get iv size
    ivSize = GetIV2BSize(parent);

    // The starting of sensitive data and data size without outer wrapper
    sensitiveData = inPrivate->buffer + integritySize + ivSize;
    dataSize      = inPrivate->size - integritySize - ivSize;

    // Unmarshal input data size
    buffer = sensitiveData;
    size   = (INT32)dataSize;
    result = UINT16_Unmarshal(&dataSizeInput, &buffer, &size);
    if(result == TPM_RC_SUCCESS)
    {
        if((dataSizeInput + sizeof(UINT16)) != dataSize)
            result = TPM_RC_SENSITIVE;
        else
        {
            // Unmarshal sensitive buffer to sensitive structure
            result = TPMT_SENSITIVE_Unmarshal(sensitive, &buffer, &size);
            if(result != TPM_RC_SUCCESS || size != 0)
            {
```

```
                result = TPM_RC_SENSITIVE;
            }
        }
    }
    return result;
}

//*** SensitiveToDuplicate()
// This function prepare the duplication blob from the sensitive area.
// The operations in this function:
//  1. marshal TPMT_SENSITIVE structure into the buffer of TPM2B_PRIVATE
//  2. apply inner wrap to the sensitive area if required
//  3. apply outer wrap if required
void SensitiveToDuplicate(
    TPMT_SENSITIVE* sensitive,    // IN: sensitive structure
    TPM2B*          name,         // IN: the name of the object
    OBJECT*         parent,       // IN: The new parent object
    TPM_ALG_ID      nameAlg,      // IN: hash algorithm in public area. It
                                  //     is passed separately because we
                                  //     only pass name, rather than the
                                  //     whole public area of the object.
    TPM2B* seed,                  // IN: the external seed. If external
                                  //     seed is provided with size of 0,
                                  //     no outer wrap should be applied
                                  //     to duplication blob.
    TPMT_SYM_DEF_OBJECT* symDef,  // IN: Symmetric key definition. If the
                                  //     symmetric key algorithm is NULL,
                                  //     no inner wrap should be applied.
    TPM2B_DATA* innerSymKey,      // IN/OUT: a symmetric key may be
                                  //     provided to encrypt the inner
                                  //     wrap of a duplication blob. May
                                  //     be generated here if needed.
    TPM2B_PRIVATE* outPrivate     // OUT: output private structure
)
{
    BYTE*         sensitiveData;             // pointer to the sensitive data
    TPMI_ALG_HASH outerHash = TPM_ALG_NULL;  // The hash algorithm for outer wrap
    TPMI_ALG_HASH innerHash = TPM_ALG_NULL;  // The hash algorithm for inner wrap
    UINT16        dataSize;                  // data blob size
    BOOL          doInnerWrap = FALSE;
    BOOL          doOuterWrap = FALSE;
    //
    // Make sure that name is provided
    pAssert(name != NULL && name->size != 0);

    // Make sure symDef and innerSymKey are not NULL
    pAssert(symDef != NULL && innerSymKey != NULL);

    // Starting of sensitive data without wrappers
    sensitiveData = outPrivate->t.buffer;

    // Find out if inner wrap is required
    if(symDef->algorithm != TPM_ALG_NULL)
    {
        doInnerWrap = TRUE;

        // Use self nameAlg as inner hash algorithm
        innerHash = nameAlg;

        // Adjust sensitive data pointer
        sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(innerHash);
    }
    // Find out if outer wrap is required
    if(seed->size != 0)
    {
        doOuterWrap = TRUE;
```

```
        // Use parent nameAlg as outer hash algorithm
        outerHash = parent->publicArea.nameAlg;

        // Adjust sensitive data pointer
        sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
    }
    // Marshal sensitive area
    dataSize = MarshalSensitive(NULL, sensitiveData, sensitive, nameAlg);

    // Apply inner wrap for duplication blob.  It includes both integrity and
    // encryption
    if(doInnerWrap)
    {
        BYTE* innerBuffer = NULL;
        BOOL  symKeyInput = TRUE;
        innerBuffer       = outPrivate->t.buffer;
        // Skip outer integrity space
        if(doOuterWrap)
            innerBuffer += sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
        dataSize = ProduceInnerIntegrity(name, innerHash, dataSize, innerBuffer);
        // Generate inner encryption key if needed
        if(innerSymKey->t.size == 0)
        {
            innerSymKey->t.size = (symDef->keyBits.sym + 7) / 8;
            CryptRandomGenerate(innerSymKey->t.size, innerSymKey->t.buffer);

            // TPM generates symmetric encryption.  Set the flag to FALSE
            symKeyInput = FALSE;
        }
        else
        {
            // assume the input key size should matches the symmetric definition
            pAssert(innerSymKey->t.size == (symDef->keyBits.sym + 7) / 8);
        }

        // Encrypt inner buffer in place
        CryptSymmetricEncrypt(innerBuffer,
                              symDef->algorithm,
                              symDef->keyBits.sym,
                              innerSymKey->t.buffer,
                              NULL,
                              TPM_ALG_CFB,
                              dataSize,
                              innerBuffer);

        // If the symmetric encryption key is imported, clear the buffer for
        // output
        if(symKeyInput)
            innerSymKey->t.size = 0;
    }
    // Apply outer wrap for duplication blob.  It includes both integrity and
    // encryption
    if(doOuterWrap)
    {
        dataSize = ProduceOuterWrap(
            parent, name, outerHash, seed, FALSE, dataSize, outPrivate->t.buffer);
    }
    // Data size for output
    outPrivate->t.size = dataSize;

    return;
}

//*** DuplicateToSensitive()
// Unwrap a duplication blob.  Check the integrity, decrypt and retrieve data
```

```
// to a sensitive structure.
// The operations in this function:
//  1. check the integrity HMAC of the input private area
//  2. decrypt the private buffer
//  3. unmarshal TPMT_SENSITIVE structure into the buffer of TPMT_SENSITIVE
//
//  Return Type: TPM_RC
//      TPM_RC_INSUFFICIENT      unmarshaling sensitive data from 'inPrivate' failed
//      TPM_RC_INTEGRITY         'inPrivate' data integrity is broken
//      TPM_RC_SIZE              unmarshaling sensitive data from 'inPrivate' failed
TPM_RC
DuplicateToSensitive(
    TPM2B*     inPrivate,          // IN: input private structure
    TPM2B*     name,               // IN: the name of the object
    OBJECT*    parent,             // IN: the parent
    TPM_ALG_ID nameAlg,            // IN: hash algorithm in public area.
    TPM2B*     seed,               // IN: an external seed may be provided.
                                   //     If external seed is provided with
                                   //     size of 0, no outer wrap is
                                   //     applied
    TPMT_SYM_DEF_OBJECT* symDef,   // IN: Symmetric key definition. If the
                                   //     symmetric key algorithm is NULL,
                                   //     no inner wrap is applied
    TPM2B* innerSymKey,            // IN: a symmetric key may be provided
                                   //     to decrypt the inner wrap of a
                                   //     duplication blob.
    TPMT_SENSITIVE* sensitive      // OUT: sensitive structure
)
{
    TPM_RC result;
    BYTE*  buffer;
    INT32  size;
    BYTE*  sensitiveData;  // pointer to the sensitive data
    UINT16 dataSize;
    UINT16 dataSizeInput;
    //
    // Make sure that name is provided
    pAssert(name != NULL && name->size != 0);

    // Make sure symDef and innerSymKey are not NULL
    pAssert(symDef != NULL && innerSymKey != NULL);

    // Starting of sensitive data
    sensitiveData = inPrivate->buffer;
    dataSize      = inPrivate->size;

    // Find out if outer wrap is applied
    if(seed->size != 0)
    {
        // Use parent nameAlg as outer hash algorithm
        TPMI_ALG_HASH outerHash = parent->publicArea.nameAlg;

        result                  = UnwrapOuter(
            parent, name, outerHash, seed, FALSE, dataSize, sensitiveData);
        if(result != TPM_RC_SUCCESS)
            return result;
        // Adjust sensitive data pointer and size
        sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
        dataSize -= sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
    }
    // Find out if inner wrap is applied
    if(symDef->algorithm != TPM_ALG_NULL)
    {
        // assume the input key size matches the symmetric definition
        pAssert(innerSymKey->size == (symDef->keyBits.sym + 7) / 8);
```

```
        // Decrypt inner buffer in place
        CryptSymmetricDecrypt(sensitiveData,
                              symDef->algorithm,
                              symDef->keyBits.sym,
                              innerSymKey->buffer,
                              NULL,
                              TPM_ALG_CFB,
                              dataSize,
                              sensitiveData);
        // Check inner integrity
        result = CheckInnerIntegrity(name, nameAlg, dataSize, sensitiveData);
        if(result != TPM_RC_SUCCESS)
            return result;
        // Adjust sensitive data pointer and size
        sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(nameAlg);
        dataSize -= sizeof(UINT16) + CryptHashGetDigestSize(nameAlg);
    }
    // Unmarshal input data size
    buffer = sensitiveData;
    size   = (INT32)dataSize;
    result = UINT16_Unmarshal(&dataSizeInput, &buffer, &size);
    if(result == TPM_RC_SUCCESS)
    {
        if((dataSizeInput + sizeof(UINT16)) != dataSize)
            result = TPM_RC_SIZE;
        else
        {
            // Unmarshal sensitive buffer to sensitive structure
            result = TPMT_SENSITIVE_Unmarshal(sensitive, &buffer, &size);

            // if the results is OK make sure that all the data was unmarshaled
            if(result == TPM_RC_SUCCESS && size != 0)
                result = TPM_RC_SIZE;
        }
    }
    return result;
}

//*** SecretToCredential()
// This function prepare the credential blob from a secret (a TPM2B_DIGEST)
// The operations in this function:
//  1. marshal TPM2B_DIGEST structure into the buffer of TPM2B_ID_OBJECT
//  2. encrypt the private buffer, excluding the leading integrity HMAC area
//  3. compute integrity HMAC and append to the beginning of the buffer.
//  4. Set the total size of TPM2B_ID_OBJECT buffer
void SecretToCredential(TPM2B_DIGEST*    secret,      // IN: secret information
                        TPM2B*           name,        // IN: the name of the object
                        TPM2B*           seed,        // IN: an external seed.
                        OBJECT*          protector,   // IN: the protector
                        TPM2B_ID_OBJECT* outIDObject  // OUT: output credential
)
{
    BYTE*         buffer;         // Auxiliary buffer pointer
    BYTE*         sensitiveData;  // pointer to the sensitive data
    TPMI_ALG_HASH outerHash;      // The hash algorithm for outer wrap
    UINT16        dataSize;       // data blob size
                                  //
    pAssert(secret != NULL && outIDObject != NULL);

    // use protector's name algorithm as outer hash ????
    outerHash = protector->publicArea.nameAlg;

    // Marshal secret area to credential buffer, leave space for integrity
    sensitiveData = outIDObject->t.credential + sizeof(UINT16)
                  + CryptHashGetDigestSize(outerHash);
    // Marshal secret area
```

```
    buffer   = sensitiveData;
    dataSize = TPM2B_DIGEST_Marshal(secret, &buffer, NULL);

    // Apply outer wrap
    outIDObject->t.size = ProduceOuterWrap(
        protector, name, outerHash, seed, FALSE, dataSize, outIDObject->t.credential);
    return;
}

//*** CredentialToSecret()
// Unwrap a credential.  Check the integrity, decrypt and retrieve data
// to a TPM2B_DIGEST structure.
// The operations in this function:
//  1. check the integrity HMAC of the input credential area
//  2. decrypt the credential buffer
//  3. unmarshal TPM2B_DIGEST structure into the buffer of TPM2B_DIGEST
//
//  Return Type: TPM_RC
//      TPM_RC_INSUFFICIENT      error during credential unmarshaling
//      TPM_RC_INTEGRITY         credential integrity is broken
//      TPM_RC_SIZE              error during credential unmarshaling
//      TPM_RC_VALUE             IV size does not match the encryption algorithm
//                               block size
TPM_RC
CredentialToSecret(TPM2B*        inIDObject,  // IN: input credential blob
                   TPM2B*        name,        // IN: the name of the object
                   TPM2B*        seed,        // IN: an external seed.
                   OBJECT*       protector,   // IN: the protector
                   TPM2B_DIGEST* secret       // OUT: secret information
)
{
    TPM_RC        result;
    BYTE*         buffer;
    INT32         size;
    TPMI_ALG_HASH outerHash;      // The hash algorithm for outer wrap
    BYTE*         sensitiveData;  // pointer to the sensitive data
    UINT16        dataSize;
    //
    // use protector's name algorithm as outer hash
    outerHash = protector->publicArea.nameAlg;

    // Unwrap outer, a TPM_RC_INTEGRITY error may be returned at this point
    result = UnwrapOuter(protector,
                         name,
                         outerHash,
                         seed,
                         FALSE,
                         inIDObject->size,
                         inIDObject->buffer);
    if(result == TPM_RC_SUCCESS)
    {
        // Compute the beginning of sensitive data
        sensitiveData =
            inIDObject->buffer + sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
        dataSize =
            inIDObject->size - (sizeof(UINT16) + CryptHashGetDigestSize(outerHash));
        // Unmarshal secret buffer to TPM2B_DIGEST structure
        buffer = sensitiveData;
        size   = (INT32)dataSize;
        result = TPM2B_DIGEST_Unmarshal(secret, &buffer, &size);

        // If there were no other unmarshaling errors, make sure that the
        // expected amount of data was recovered
        if(result == TPM_RC_SUCCESS && size != 0)
            return TPM_RC_SIZE;
    }
```

```
    return result;
}

//*** MemoryRemoveTrailingZeros()
// This function is used to adjust the length of an authorization value.
// It adjusts the size of the TPM2B so that it does not include octets
// at the end of the buffer that contain zero.
// The function returns the number of non-zero octets in the buffer.
UINT16
MemoryRemoveTrailingZeros(TPM2B_AUTH* auth  // IN/OUT: value to adjust
)
{
    while((auth->t.size > 0) && (auth->t.buffer[auth->t.size - 1] == 0))
        auth->t.size--;
    return auth->t.size;
}

//*** SetLabelAndContext()
// This function sets the label and context for a derived key. It is possible
// that 'label' or 'context' can end up being an Empty Buffer.
TPM_RC
SetLabelAndContext(TPMS_DERIVE* labelContext,       // IN/OUT: the recovered label and
                                                    //      context
                   TPM2B_SENSITIVE_DATA* sensitive  // IN: the sensitive data
)
{
    TPMS_DERIVE sensitiveValue;
    TPM_RC      result;
    INT32       size;
    BYTE*       buff;
    //
    // Unmarshal a TPMS_DERIVE from the TPM2B_SENSITIVE_DATA buffer
    // If there is something to unmarshal...
    if(sensitive->t.size != 0)
    {
        size   = sensitive->t.size;
        buff   = sensitive->t.buffer;
        result = TPMS_DERIVE_Unmarshal(&sensitiveValue, &buff, &size);
        if(result != TPM_RC_SUCCESS)
            return result;
        // If there was a label in the public area leave it there, otherwise, copy
        // the new value
        if(labelContext->label.t.size == 0)
            MemoryCopy2B(&labelContext->label.b,
                         &sensitiveValue.label.b,
                         sizeof(labelContext->label.t.buffer));
        // if there was a context string in publicArea, it overrides
        if(labelContext->context.t.size == 0)
            MemoryCopy2B(&labelContext->context.b,
                         &sensitiveValue.context.b,
                         sizeof(labelContext->label.t.buffer));
    }
    return TPM_RC_SUCCESS;
}

//*** UnmarshalToPublic()
// Support function to unmarshal the template. This is used because the
// Input may be a TPMT_TEMPLATE and that structure does not have the same
// size as a TPMT_PUBLIC because of the difference between the 'unique' and
// 'seed' fields.
// If 'derive' is not NULL, then the 'seed' field is assumed to contain
// a 'label' and 'context' that are unmarshaled into 'derive'.
TPM_RC
UnmarshalToPublic(TPMT_PUBLIC*    tOut,  // OUT: output
                  TPM2B_TEMPLATE* tIn,   // IN:
                  BOOL derivation,       // IN: indicates if this is for a derivation
```

```
                TPMS_DERIVE* labelContext  // OUT: label and context if derivation
)
{
    BYTE*  buffer = tIn->t.buffer;
    INT32  size   = tIn->t.size;
    TPM_RC result;
    //
    // make sure that tOut is zeroed so that there are no remnants from previous
    // uses
    MemorySet(tOut, 0, sizeof(TPMT_PUBLIC));
    // Unmarshal the components of the TPMT_PUBLIC up to the unique field
    result = TPMI_ALG_PUBLIC_Unmarshal(&tOut->type, &buffer, &size);
    if(result != TPM_RC_SUCCESS)
        return result;
    result = TPMI_ALG_HASH_Unmarshal(&tOut->nameAlg, &buffer, &size, FALSE);
    if(result != TPM_RC_SUCCESS)
        return result;
    result = TPMA_OBJECT_Unmarshal(&tOut->objectAttributes, &buffer, &size);
    if(result != TPM_RC_SUCCESS)
        return result;
    result = TPM2B_DIGEST_Unmarshal(&tOut->authPolicy, &buffer, &size);
    if(result != TPM_RC_SUCCESS)
        return result;
    result =
        TPMU_PUBLIC_PARMS_Unmarshal(&tOut->parameters, &buffer, &size, tOut->type);
    if(result != TPM_RC_SUCCESS)
        return result;
    // Now unmarshal a TPMS_DERIVE if this is for derivation
    if(derivation)
        result = TPMS_DERIVE_Unmarshal(labelContext, &buffer, &size);
    else
        // otherwise, unmarshal a TPMU_PUBLIC_ID
        result = TPMU_PUBLIC_ID_Unmarshal(&tOut->unique, &buffer, &size, tOut->type);
    // Make sure the template was used up
    if((result == TPM_RC_SUCCESS) && (size != 0))
        result = TPM_RC_SIZE;
    return result;
}

//*** ObjectSetExternal()
// Set the external attributes for an object.
void ObjectSetExternal(OBJECT* object)
{
    object->attributes.external = SET;
}
```

7.109 /tpm/src/command/Object/ReadPublic.c

```
#include "Tpm.h"
#include "ReadPublic_fp.h"

#if CC_ReadPublic  // Conditional expansion of this file

/*(See part 3 specification)
// read public area of a loaded object
*/
//  Return Type: TPM_RC
//      TPM_RC_SEQUENCE             can not read the public area of a sequence
//                                  object
TPM_RC
TPM2_ReadPublic(ReadPublic_In*  in,  // IN: input parameter list
                ReadPublic_Out* out  // OUT: output parameter list
)
{
    OBJECT* object = HandleToObject(in->objectHandle);
```

```
    // Input Validation
    // Can not read public area of a sequence object
    if(ObjectIsSequence(object))
        return TPM_RC_SEQUENCE;

    // Command Output
    out->outPublic.publicArea = object->publicArea;
    out->name                 = object->name;
    out->qualifiedName        = object->qualifiedName;

    return TPM_RC_SUCCESS;
}

#endif  // CC_ReadPublic
```

7.110 /tpm/src/command/Object/Unseal.c

```
#include "Tpm.h"
#include "Unseal_fp.h"

#if CC_Unseal  // Conditional expansion of this file

/*(See part 3 specification)
// return data in a sealed data blob
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES         'itemHandle' has wrong attributes
//      TPM_RC_TYPE               'itemHandle' is not a KEYEDHASH data object
TPM_RC
TPM2_Unseal(Unseal_In* in, Unseal_Out* out)
{
    OBJECT* object;
    // Input Validation
    // Get pointer to loaded object
    object = HandleToObject(in->itemHandle);

    // Input handle must be a data object
    if(object->publicArea.type != TPM_ALG_KEYEDHASH)
        return TPM_RCS_TYPE + RC_Unseal_itemHandle;
    if(IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, decrypt)
       || IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, sign)
       || IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, restricted))
        return TPM_RCS_ATTRIBUTES + RC_Unseal_itemHandle;
    // Command Output
    // Copy data
    out->outData = object->sensitive.sensitive.bits;
    return TPM_RC_SUCCESS;
}

#endif  // CC_Unseal
```

7.111 /tpm/src/command/PCR/PCR_Allocate.c

```
#include "Tpm.h"
#include "PCR_Allocate_fp.h"

#if CC_PCR_Allocate  // Conditional expansion of this file

/*(See part 3 specification)
// Allocate PCR banks
*/
//  Return Type: TPM_RC
//      TPM_RC_PCR              the allocation did not have required PCR
```

```
//      TPM_RC_NV_UNAVAILABLE   NV is not accessible
//      TPM_RC_NV_RATE          NV is in a rate-limiting mode
TPM_RC
TPM2_PCR_Allocate(PCR_Allocate_In*  in,  // IN: input parameter list
                  PCR_Allocate_Out* out  // OUT: output parameter list
)
{
    TPM_RC result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point.
    // Note: These codes are not listed in the return values above because it is
    // an implementation choice to check in this routine rather than in a common
    // function that is called before these actions are called. These return values
    // are described in the Response Code section of Part 3.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Command Output

    // Call PCR Allocation function.
    result = PCRAllocate(
        &in->pcrAllocation, &out->maxPCR, &out->sizeNeeded, &out->sizeAvailable);
    if(result == TPM_RC_PCR)
        return result;

    //
    out->allocationSuccess = (result == TPM_RC_SUCCESS);

    // if re-configuration succeeds, set the flag to indicate PCR configuration is
    // going to be changed in next boot
    if(out->allocationSuccess == YES)
        g_pcrReConfig = TRUE;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PCR_Allocate
```

7.112 /tpm/src/command/PCR/PCR_Event.c

```
#include "Tpm.h"
#include "PCR_Event_fp.h"

#if CC_PCR_Event  // Conditional expansion of this file

/*(See part 3 specification)
// Update PCR
*/
//  Return Type: TPM_RC
//      TPM_RC_LOCALITY             current command locality is not allowed to
//                                  extend the PCR referenced by 'pcrHandle'
TPM_RC
TPM2_PCR_Event(PCR_Event_In*  in,  // IN: input parameter list
               PCR_Event_Out* out  // OUT: output parameter list
)
{
    HASH_STATE hashState;
    UINT32     i;
    UINT16     size;

    // Input Validation

    // If a PCR extend is required
    if(in->pcrHandle != TPM_RH_NULL)
```

```
    {
        // If the PCR is not allow to extend, return error
        if(!PCRIsExtendAllowed(in->pcrHandle))
            return TPM_RC_LOCALITY;

        // If PCR is state saved and we need to update orderlyState, check NV
        // availability
        if(PCRIsStateSaved(in->pcrHandle))
            RETURN_IF_ORDERLY;
    }

    // Internal Data Update

    out->digests.count = HASH_COUNT;

    // Iterate supported PCR bank algorithms to extend
    for(i = 0; i < HASH_COUNT; i++)
    {
        TPM_ALG_ID hash                 = CryptHashGetAlgByIndex(i);
        out->digests.digests[i].hashAlg = hash;
        size                            = CryptHashStart(&hashState, hash);
        CryptDigestUpdate2B(&hashState, &in->eventData.b);
        CryptHashEnd(&hashState, size, (BYTE*)&out->digests.digests[i].digest);
        if(in->pcrHandle != TPM_RH_NULL)
            PCRExtend(
                in->pcrHandle, hash, size, (BYTE*)&out->digests.digests[i].digest);
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_PCR_Event
```

7.113 /tpm/src/command/PCR/PCR_Extend.c

```
#include "Tpm.h"
#include "PCR_Extend_fp.h"

#if CC_PCR_Extend  // Conditional expansion of this file

/*(See part 3 specification)
// Update PCR
*/
//  Return Type: TPM_RC
//      TPM_RC_LOCALITY             current command locality is not allowed to
//                                  extend the PCR referenced by 'pcrHandle'
TPM_RC
TPM2_PCR_Extend(PCR_Extend_In* in  // IN: input parameter list
)
{
    UINT32 i;

    // Input Validation

    // NOTE: This function assumes that the unmarshaling function for 'digests' will
    // have validated that all of the indicated hash algorithms are valid. If the
    // hash algorithms are correct, the unmarshaling code will unmarshal a digest
    // of the size indicated by the hash algorithm. If the overall size is not
    // consistent, the unmarshaling code will run out of input data or have input
    // data left over. In either case, it will cause an unmarshaling error and this
    // function will not be called.

    // For NULL handle, do nothing and return success
    if(in->pcrHandle == TPM_RH_NULL)
        return TPM_RC_SUCCESS;
```

```
    // Check if the extend operation is allowed by the current command locality
    if(!PCRIsExtendAllowed(in->pcrHandle))
        return TPM_RC_LOCALITY;

    // If PCR is state saved and we need to update orderlyState, check NV
    // availability
    if(PCRIsStateSaved(in->pcrHandle))
        RETURN_IF_ORDERLY;

    // Internal Data Update

    // Iterate input digest list to extend
    for(i = 0; i < in->digests.count; i++)
    {
        PCRExtend(in->pcrHandle,
                  in->digests.digests[i].hashAlg,
                  CryptHashGetDigestSize(in->digests.digests[i].hashAlg),
                  (BYTE*)&in->digests.digests[i].digest);
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_PCR_Extend
```

7.114 /tpm/src/command/PCR/PCR_Read.c

```
#include "Tpm.h"
#include "PCR_Read_fp.h"

#if CC_PCR_Read  // Conditional expansion of this file

/*(See part 3 specification)
// Read a set of PCR
*/
TPM_RC
TPM2_PCR_Read(PCR_Read_In*  in,  // IN: input parameter list
              PCR_Read_Out* out  // OUT: output parameter list
)
{
    // Command Output

    // Call PCR read function.  input pcrSelectionIn parameter could be changed
    // to reflect the actual PCR being returned
    PCRRead(&in->pcrSelectionIn, &out->pcrValues, &out->pcrUpdateCounter);

    out->pcrSelectionOut = in->pcrSelectionIn;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PCR_Read
```

7.115 /tpm/src/command/PCR/PCR_Reset.c

```
#include "Tpm.h"
#include "PCR_Reset_fp.h"

#if CC_PCR_Reset  // Conditional expansion of this file

/*(See part 3 specification)
// Reset PCR
*/
```

```
//  Return Type: TPM_RC
//      TPM_RC_LOCALITY             current command locality is not allowed to
//                                  reset the PCR referenced by 'pcrHandle'
TPM_RC
TPM2_PCR_Reset(PCR_Reset_In* in  // IN: input parameter list
)
{
    // Input Validation

    // Check if the reset operation is allowed by the current command locality
    if(!PCRIsResetAllowed(in->pcrHandle))
        return TPM_RC_LOCALITY;

    // If PCR is state saved and we need to update orderlyState, check NV
    // availability
    if(PCRIsStateSaved(in->pcrHandle))
        RETURN_IF_ORDERLY;

    // Internal Data Update

    // Reset selected PCR in all banks to 0
    PCRSetValue(in->pcrHandle, 0);

    // Indicate that the PCR changed so that pcrCounter will be incremented if
    // necessary.
    PCRChanged(in->pcrHandle);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PCR_Reset
```

7.116 /tpm/src/command/PCR/PCR_SetAuthPolicy.c

```
#include "Tpm.h"
#include "PCR_SetAuthPolicy_fp.h"

#if CC_PCR_SetAuthPolicy  // Conditional expansion of this file

/*(See part 3 specification)
// Set authPolicy to a group of PCR
*/
//  Return Type: TPM_RC
//      TPM_RC_SIZE                 size of 'authPolicy' is not the size of a digest
//                                  produced by 'policyDigest'
//      TPM_RC_VALUE                PCR referenced by 'pcrNum' is not a member
//                                  of a PCR policy group
TPM_RC
TPM2_PCR_SetAuthPolicy(PCR_SetAuthPolicy_In* in  // IN: input parameter list
)
{
    UINT32 groupIndex;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Input Validation:

    // Check the authPolicy consistent with hash algorithm
    if(in->authPolicy.t.size != CryptHashGetDigestSize(in->hashAlg))
        return TPM_RCS_SIZE + RC_PCR_SetAuthPolicy_authPolicy;

    // If PCR does not belong to a policy group, return TPM_RC_VALUE
```

```
    if(!PCRBelongsPolicyGroup(in->pcrNum, &groupIndex))
        return TPM_RCS_VALUE + RC_PCR_SetAuthPolicy_pcrNum;

    // Internal Data Update

    // Set PCR policy
    gp.pcrPolicies.hashAlg[groupIndex] = in->hashAlg;
    gp.pcrPolicies.policy[groupIndex]  = in->authPolicy;

    // Save new policy to NV
    NV_SYNC_PERSISTENT(pcrPolicies);

    return TPM_RC_SUCCESS;
}

#endif  // CC_PCR_SetAuthPolicy
```

7.117 /tpm/src/command/PCR/PCR_SetAuthValue.c

```
#include "Tpm.h"
#include "PCR_SetAuthValue_fp.h"

#if CC_PCR_SetAuthValue  // Conditional expansion of this file

/*(See part 3 specification)
// Set authValue to a group of PCR
*/
//  Return Type: TPM_RC
//      TPM_RC_VALUE                PCR referenced by 'pcrHandle' is not a member
//                                  of a PCR authorization group
TPM_RC
TPM2_PCR_SetAuthValue(PCR_SetAuthValue_In* in  // IN: input parameter list
)
{
    UINT32 groupIndex;
    // Input Validation:

    // If PCR does not belong to an auth group, return TPM_RC_VALUE
    if(!PCRBelongsAuthGroup(in->pcrHandle, &groupIndex))
        return TPM_RC_VALUE;

    // The command may cause the orderlyState to be cleared due to the update of
    // state clear data.  If this is the case, Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_ORDERLY;

    // Internal Data Update

    // Set PCR authValue
    MemoryRemoveTrailingZeros(&in->auth);
    gc.pcrAuthValues.auth[groupIndex] = in->auth;

    return TPM_RC_SUCCESS;
}

#endif  // CC_PCR_SetAuthValue
```

7.118 /tpm/src/command/Random/GetRandom.c

```
#include "Tpm.h"
#include "GetRandom_fp.h"

#if CC_GetRandom  // Conditional expansion of this file
```

```
/*(See part 3 specification)
// random number generator
*/
TPM_RC
TPM2_GetRandom(GetRandom_In*  in,  // IN: input parameter list
               GetRandom_Out* out  // OUT: output parameter list
)
{
    // Command Output

    // if the requested bytes exceed the output buffer size, generates the
    // maximum bytes that the output buffer allows
    if(in->bytesRequested > sizeof(TPMU_HA))
        out->randomBytes.t.size = sizeof(TPMU_HA);
    else
        out->randomBytes.t.size = in->bytesRequested;

    CryptRandomGenerate(out->randomBytes.t.size, out->randomBytes.t.buffer);

    return TPM_RC_SUCCESS;
}

#endif  // CC_GetRandom
```

7.119 /tpm/src/command/Random/StirRandom.c

```
#include "Tpm.h"
#include "StirRandom_fp.h"

#if CC_StirRandom  // Conditional expansion of this file

/*(See part 3 specification)
// add entropy to the RNG state
*/
TPM_RC
TPM2_StirRandom(StirRandom_In* in  // IN: input parameter list
)
{
    // Internal Data Update
    CryptRandomStir(in->inData.t.size, in->inData.t.buffer);

    return TPM_RC_SUCCESS;
}

#endif  // CC_StirRandom
```

7.120 /tpm/src/command/Session/PolicyRestart.c

```
#include "Tpm.h"
#include "PolicyRestart_fp.h"

#if CC_PolicyRestart  // Conditional expansion of this file

/*(See part 3 specification)
// Restore a policy session to its initial state
*/
TPM_RC
TPM2_PolicyRestart(PolicyRestart_In* in  // IN: input parameter list
)
{
    // Initialize policy session data
    SessionResetPolicyData(SessionGet(in->sessionHandle));
```

```
    return TPM_RC_SUCCESS;
}

#endif  // CC_PolicyRestart
```

7.121 /tpm/src/command/Session/StartAuthSession.c

```
#include "Tpm.h"
#include "StartAuthSession_fp.h"

#if CC_StartAuthSession  // Conditional expansion of this file

/*(See part 3 specification)
// Start an authorization session
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       'tpmKey' does not reference a decrypt key
//      TPM_RC_CONTEXT_GAP      the difference between the most recently created
//                              active context and the oldest active context is at
//                              the limits of the TPM
//      TPM_RC_HANDLE           input decrypt key handle only has public portion
//                              loaded
//      TPM_RC_MODE             'symmetric' specifies a block cipher but the mode
//                              is not TPM_ALG_CFB.
//      TPM_RC_SESSION_HANDLES  no session handle is available
//      TPM_RC_SESSION_MEMORY   no more slots for loading a session
//      TPM_RC_SIZE             nonce less than 16 octets or greater than the size
//                              of the digest produced by 'authHash'
//      TPM_RC_VALUE            secret size does not match decrypt key type; or the
//                              recovered secret is larger than the digest size of
//                              the nameAlg of 'tpmKey'; or, for an RSA decrypt key,
//                              if 'encryptedSecret' is greater than the
//                              public modulus of 'tpmKey'.
TPM_RC
TPM2_StartAuthSession(StartAuthSession_In*  in,  // IN: input parameter buffer
                      StartAuthSession_Out* out  // OUT: output parameter buffer
)
{
    TPM_RC     result = TPM_RC_SUCCESS;
    OBJECT*    tpmKey;  // TPM key for decrypt salt
    TPM2B_DATA salt;

    // Input Validation

    // Check input nonce size.  IT should be at least 16 bytes but not larger
    // than the digest size of session hash.
    if(in->nonceCaller.t.size < 16
       || in->nonceCaller.t.size > CryptHashGetDigestSize(in->authHash))
        return TPM_RCS_SIZE + RC_StartAuthSession_nonceCaller;

    // If an decrypt key is passed in, check its validation
    if(in->tpmKey != TPM_RH_NULL)
    {
        // Get pointer to loaded decrypt key
        tpmKey = HandleToObject(in->tpmKey);

        // key must be asymmetric with its sensitive area loaded. Since this
        // command does not require authorization, the presence of the sensitive
        // area was not already checked as it is with most other commands that
        // use the sensitive are so check it here
        if(!CryptIsAsymAlgorithm(tpmKey->publicArea.type))
            return TPM_RCS_KEY + RC_StartAuthSession_tpmKey;
        // secret size cannot be 0
        if(in->encryptedSalt.t.size == 0)
            return TPM_RCS_VALUE + RC_StartAuthSession_encryptedSalt;
```

```
        // Decrypting salt requires accessing the private portion of a key.
        // Therefore, tmpKey can not be a key with only public portion loaded
        if(tpmKey->attributes.publicOnly)
            return TPM_RCS_HANDLE + RC_StartAuthSession_tpmKey;
        // HMAC session input handle check.
        // tpmKey should be a decryption key
        if(!IS_ATTRIBUTE(tpmKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
            return TPM_RCS_ATTRIBUTES + RC_StartAuthSession_tpmKey;
        // Secret Decryption.  A TPM_RC_VALUE, TPM_RC_KEY or Unmarshal errors
        // may be returned at this point
        result = CryptSecretDecrypt(
            tpmKey, &in->nonceCaller, SECRET_KEY, &in->encryptedSalt, &salt);
        if(result != TPM_RC_SUCCESS)
            return TPM_RCS_VALUE + RC_StartAuthSession_encryptedSalt;
    }
    else
    {
        // secret size must be 0
        if(in->encryptedSalt.t.size != 0)
            return TPM_RCS_VALUE + RC_StartAuthSession_encryptedSalt;
        salt.t.size = 0;
    }
    switch(HandleGetType(in->bind))
    {
        case TPM_HT_TRANSIENT:
        {
            OBJECT* object = HandleToObject(in->bind);
            // If the bind handle references a transient object, make sure that we
            // can get to the authorization value. Also, make sure that the object
            // has a proper Name (nameAlg != TPM_ALG_NULL). If it doesn't, then
            // it might be possible to bind to an object where the authValue is
            // known. This does not create a real issue in that, if you know the
            // authorization value, you can actually bind to the object. However,
            // there is a potential
            if(object->attributes.publicOnly == SET)
                return TPM_RCS_HANDLE + RC_StartAuthSession_bind;
            break;
        }
        case TPM_HT_NV_INDEX:
            // a PIN index can't be a bind object
            {
                NV_INDEX* nvIndex = NvGetIndexInfo(in->bind, NULL);
                if(IsNvPinPassIndex(nvIndex->publicArea.attributes)
                   || IsNvPinFailIndex(nvIndex->publicArea.attributes))
                    return TPM_RCS_HANDLE + RC_StartAuthSession_bind;
                break;
            }
        default:
            break;
    }
    // If 'symmetric' is a symmetric block cipher (not TPM_ALG_NULL or TPM_ALG_XOR)
    // then the mode must be CFB.
    if(in->symmetric.algorithm != TPM_ALG_NULL
       && in->symmetric.algorithm != TPM_ALG_XOR
       && in->symmetric.mode.sym != TPM_ALG_CFB)
        return TPM_RCS_MODE + RC_StartAuthSession_symmetric;

    // Internal Data Update and command output

    // Create internal session structure.  TPM_RC_CONTEXT_GAP, TPM_RC_NO_HANDLES
    // or TPM_RC_SESSION_MEMORY errors may be returned at this point.
    //
    // The detailed actions for creating the session context are not shown here
    // as the details are implementation dependent
    // SessionCreate sets the output handle and nonceTPM
    result = SessionCreate(in->sessionType,
```

```
                            in->authHash,
                            &in->nonceCaller,
                            &in->symmetric,
                            in->bind,
                            &salt,
                            &out->sessionHandle,
                            &out->nonceTPM);
    return result;
}

#endif  // CC_StartAuthSession
```

7.122 /tpm/src/command/Signature/Sign.c

```
#include "Tpm.h"
#include "Sign_fp.h"

#if CC_Sign  // Conditional expansion of this file

#  include "Attest_spt_fp.h"

/*(See part 3 specification)
// sign an externally provided hash using an asymmetric signing key
*/
//  Return Type: TPM_RC
//      TPM_RC_BINDING          The public and private portions of the key are not
//                              properly bound.
//      TPM_RC_KEY              'signHandle' does not reference a signing key;
//      TPM_RC_SCHEME           the scheme is not compatible with sign key type,
//                              or input scheme is not compatible with default
//                              scheme, or the chosen scheme is not a valid
//                              sign scheme
//      TPM_RC_TICKET           'validation' is not a valid ticket
//      TPM_RC_VALUE            the value to sign is larger than allowed for the
//                              type of 'keyHandle'

TPM_RC
TPM2_Sign(Sign_In*  in,  // IN: input parameter list
          Sign_Out* out  // OUT: output parameter list
)
{
    TPM_RC            result;
    TPMT_TK_HASHCHECK ticket;
    OBJECT*           signObject = HandleToObject(in->keyHandle);
    //
    // Input Validation
    if(!IsSigningObject(signObject))
        return TPM_RCS_KEY + RC_Sign_keyHandle;

    // A key that will be used for x.509 signatures can't be used in TPM2_Sign().
    if(IS_ATTRIBUTE(signObject->publicArea.objectAttributes, TPMA_OBJECT, x509sign))
        return TPM_RCS_ATTRIBUTES + RC_Sign_keyHandle;

    // pick a scheme for sign.  If the input sign scheme is not compatible with
    // the default scheme, return an error.
    if(!CryptSelectSignScheme(signObject, &in->inScheme))
        return TPM_RCS_SCHEME + RC_Sign_inScheme;

    // If validation is provided, or the key is restricted, check the ticket
    if(in->validation.digest.t.size != 0
       || IS_ATTRIBUTE(
           signObject->publicArea.objectAttributes, TPMA_OBJECT, restricted))
    {
        // Compute and compare ticket
        result = TicketComputeHashCheck(in->validation.hierarchy,
```

```
                                        in->inScheme.details.any.hashAlg,
                                        &in->digest,
                                        &ticket);
        if(result != TPM_RC_SUCCESS)
            return result;

        if(!MemoryEqual2B(&in->validation.digest.b, &ticket.digest.b))
            return TPM_RCS_TICKET + RC_Sign_validation;
    }
    else
    // If we don't have a ticket, at least verify that the provided 'digest'
    // is the size of the scheme hashAlg digest.
    // NOTE: this does not guarantee that the 'digest' is actually produced using
    // the indicated hash algorithm, but at least it might be.
    {
        if(in->digest.t.size
           != CryptHashGetDigestSize(in->inScheme.details.any.hashAlg))
            return TPM_RCS_SIZE + RC_Sign_digest;
    }

    // Command Output
    // Sign the hash. A TPM_RC_VALUE or TPM_RC_SCHEME
    // error may be returned at this point
    result = CryptSign(signObject, &in->inScheme, &in->digest, &out->signature);

    return result;
}

#endif  // CC_Sign
```

7.123 /tpm/src/command/Signature/VerifySignature.c

```
#include "Tpm.h"
#include "VerifySignature_fp.h"

#if CC_VerifySignature  // Conditional expansion of this file

/*(See part 3 specification)
// This command uses loaded key to validate an asymmetric signature on a message
// with the message digest passed to the TPM.
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES         'keyHandle' does not reference a signing key
//      TPM_RC_SIGNATURE          signature is not genuine
//      TPM_RC_SCHEME             CryptValidateSignature()
//      TPM_RC_HANDLE             the input handle is references an HMAC key but
//                                the private portion is not loaded
TPM_RC
TPM2_VerifySignature(VerifySignature_In*  in,  // IN: input parameter list
                     VerifySignature_Out* out  // OUT: output parameter list
)
{
    TPM_RC            result;
    OBJECT*           signObject = HandleToObject(in->keyHandle);
    TPMI_RH_HIERARCHY hierarchy;

    // Input Validation
    // The object to validate the signature must be a signing key.
    if(!IS_ATTRIBUTE(signObject->publicArea.objectAttributes, TPMA_OBJECT, sign))
        return TPM_RCS_ATTRIBUTES + RC_VerifySignature_keyHandle;

    // Validate Signature.  TPM_RC_SCHEME, TPM_RC_HANDLE or TPM_RC_SIGNATURE
    // error may be returned by CryptCVerifySignatrue()
    result = CryptValidateSignature(in->keyHandle, &in->digest, &in->signature);
    if(result != TPM_RC_SUCCESS)
```

```
        return RcSafeAddToResult(result, RC_VerifySignature_signature);

    // Command Output

    hierarchy = GetHierarchy(in->keyHandle);
    if(hierarchy == TPM_RH_NULL || signObject->publicArea.nameAlg == TPM_ALG_NULL)
    {
        // produce empty ticket if hierarchy is TPM_RH_NULL or nameAlg is
        // TPM_ALG_NULL
        out->validation.tag           = TPM_ST_VERIFIED;
        out->validation.hierarchy     = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else
    {
        // Compute ticket
        result = TicketComputeVerified(
            hierarchy, &in->digest, &signObject->name, &out->validation);
        if(result != TPM_RC_SUCCESS)
            return result;
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_VerifySignature
```

7.124 /tpm/src/command/Startup/Shutdown.c

```
#include "Tpm.h"
#include "Shutdown_fp.h"

#if CC_Shutdown  // Conditional expansion of this file

/*(See part 3 specification)
// Shut down TPM for power off
*/
//  Return Type: TPM_RC
//      TPM_RC_TYPE             if PCR bank has been re-configured, a
//                              Shutdown(CLEAR) is required
TPM_RC
TPM2_Shutdown(Shutdown_In* in  // IN: input parameter list
)
{
    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Input Validation
    // If PCR bank has been reconfigured, a CLEAR state save is required
    if(g_pcrReConfig && in->shutdownType == TPM_SU_STATE)
        return TPM_RCS_TYPE + RC_Shutdown_shutdownType;
    // Internal Data Update
    gp.orderlyState = in->shutdownType;

#  if USE_DA_USED
    // CLEAR g_daUsed so that any future DA-protected access will cause the
    // shutdown to become non-orderly. It is not sufficient to invalidate the
    // shutdown state after a DA failure because an attacker can inhibit access
    // to NV and use the fact that an update of failedTries was attempted as an
    // indication of an authorization failure. By making sure that the orderly state
    // is CLEAR before any DA attempt, this prevents the possibility of this 'attack.'
    g_daUsed = FALSE;
#  endif
```

```
    // PCR private date state save
    PCRStateSave(in->shutdownType);

#   if ACT_SUPPORT
    // Save the ACT state
    ActShutdown(in->shutdownType);
#   endif

    // Save RAM backed NV index data
    NvUpdateIndexOrderlyData();

#   if ACCUMULATE_SELF_HEAL_TIMER
    // Save the current time value
    go.time = g_time;
#   endif

    // Save all orderly data
    NvWrite(NV_ORDERLY_DATA, sizeof(ORDERLY_DATA), &go);

    if(in->shutdownType == TPM_SU_STATE)
    {
        // Save STATE_RESET and STATE_CLEAR data
        NvWrite(NV_STATE_CLEAR_DATA, sizeof(STATE_CLEAR_DATA), &gc);
        NvWrite(NV_STATE_RESET_DATA, sizeof(STATE_RESET_DATA), &gr);

        // Save the startup flags for resume
        if(g_DrtmPreStartup)
            gp.orderlyState = TPM_SU_STATE | PRE_STARTUP_FLAG;
        else if(g_StartupLocality3)
            gp.orderlyState = TPM_SU_STATE | STARTUP_LOCALITY_3;
    }
    // only two shutdown options.
    else if(in->shutdownType != TPM_SU_CLEAR)
        return TPM_RCS_VALUE + RC_Shutdown_shutdownType;

    NV_SYNC_PERSISTENT(orderlyState);

    return TPM_RC_SUCCESS;
}
#endif  // CC_Shutdown
```

7.125 /tpm/src/command/Startup/Startup.c

```
#include "Tpm.h"
#include "Startup_fp.h"

#if CC_Startup  // Conditional expansion of this file

/*(See part 3 specification)
// Initialize TPM because a system-wide reset
*/
//  Return Type: TPM_RC
//      TPM_RC_LOCALITY             a Startup(STATE) does not have the same H-CRTM
//                                  state as the previous Startup() or the locality
//                                  of the startup is not 0 or 3
//      TPM_RC_NV_UNINITIALIZED     the saved state cannot be recovered and a
//                                  Startup(CLEAR) is required.
//      TPM_RC_VALUE                'startup' type is not compatible with previous
//                                  shutdown sequence

TPM_RC
TPM2_Startup(Startup_In* in  // IN: input parameter list
)
{
```

```
    STARTUP_TYPE startup;
    BYTE         locality = _plat__LocalityGet();
    BOOL         OK       = TRUE;
    //
    // The command needs NV update.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Get the flags for the current startup locality and the H-CRTM.
    // Rather than generalizing the locality setting, this code takes advantage
    // of the fact that the PC Client specification only allows Startup()
    // from locality 0 and 3. To generalize this probably would require a
    // redo of the NV space and since this is a feature that is hardly ever used
    // outside of the PC Client, this code just support the PC Client needs.

    // Input Validation
    // Check that the locality is a supported value
    if(locality != 0 && locality != 3)
        return TPM_RC_LOCALITY;
    // If there was a H-CRTM, then treat the locality as being 3
    // regardless of what the Startup() was. This is done to preserve the
    // H-CRTM PCR so that they don't get overwritten with the normal
    // PCR startup initialization. This basically means that g_StartupLocality3
    // and g_DrtmPreStartup can't both be SET at the same time.
    if(g_DrtmPreStartup)
        locality = 0;
    g_StartupLocality3 = (locality == 3);

#   if USE_DA_USED
    // If there was no orderly shutdown, then there might have been a write to
    // failedTries that didn't get recorded but only if g_daUsed was SET in the
    // shutdown state
    g_daUsed = (gp.orderlyState == SU_DA_USED_VALUE);
    if(g_daUsed)
        gp.orderlyState = SU_NONE_VALUE;
#   endif

    g_prevOrderlyState = gp.orderlyState;

    // If there was a proper shutdown, then the startup modifiers are in the
    // orderlyState. Turn them off in the copy.
    if(IS_ORDERLY(g_prevOrderlyState))
        g_prevOrderlyState &= ~(PRE_STARTUP_FLAG | STARTUP_LOCALITY_3);
    // If this is a Resume,
    if(in->startupType == TPM_SU_STATE)
    {
        // then there must have been a prior TPM2_ShutdownState(STATE)
        if(g_prevOrderlyState != TPM_SU_STATE)
            return TPM_RCS_VALUE + RC_Startup_startupType;
        // and the part of NV used for state save must have been recovered
        // correctly.
        // NOTE: if this fails, then the caller will need to do Startup(CLEAR). The
        // code for Startup(Clear) cannot fail if the NV can't be read correctly
        // because that would prevent the TPM from ever getting unstuck.
        if(g_nvOk == FALSE)
            return TPM_RC_NV_UNINITIALIZED;
        // For Resume, the H-CRTM has to be the same as the previous boot
        if(g_DrtmPreStartup != ((gp.orderlyState & PRE_STARTUP_FLAG) != 0))
            return TPM_RCS_VALUE + RC_Startup_startupType;
        if(g_StartupLocality3 != ((gp.orderlyState & STARTUP_LOCALITY_3) != 0))
            return TPM_RC_LOCALITY;
    }
    // Clean up the gp state
    gp.orderlyState = g_prevOrderlyState;

    // Internal Date Update
    if((gp.orderlyState == TPM_SU_STATE) && (g_nvOk == TRUE))
```

```
    {
        // Always read the data that is only cleared on a Reset because this is not
        // a reset
        NvRead(&gr, NV_STATE_RESET_DATA, sizeof(gr));
        if(in->startupType == TPM_SU_STATE)
        {
            // If this is a startup STATE (a Resume) need to read the data
            // that is cleared on a startup CLEAR because this is not a Reset
            // or Restart.
            NvRead(&gc, NV_STATE_CLEAR_DATA, sizeof(gc));
            startup = SU_RESUME;
        }
        else
            startup = SU_RESTART;
    }
    else
        // Will do a TPM reset if Shutdown(CLEAR) and Startup(CLEAR) or no shutdown
        // or there was a failure reading the NV data.
        startup = SU_RESET;
    // Startup for cryptographic library. Don't do this until after the orderly
    // state has been read in from NV.
    OK = OK && CryptStartup(startup);

    // When the cryptographic library has been started, indicate that a TPM2_Startup
    // command has been received.
    OK = OK && TPMRegisterStartup();

#   if VENDOR_PERMANENT_AUTH_ENABLED == YES
    // Read the platform unique value that is used as VENDOR_PERMANENT_AUTH_HANDLE
    // authorization value
    g_platformUniqueAuth.t.size = (UINT16)_plat__GetUniqueAuth(
        1, sizeof(g_platformUniqueAuth.t.buffer), g_platformUniqueAuth.t.buffer);
#   endif

    // Start up subsystems
    // Start set the safe flag
    OK = OK && TimeStartup(startup);

    // Start dictionary attack subsystem
    OK = OK && DAStartup(startup);

    // Enable hierarchies
    OK = OK && HierarchyStartup(startup);

    // Restore/Initialize PCR
    OK = OK && PCRStartup(startup, locality);

    // Restore/Initialize command audit information
    OK = OK && CommandAuditStartup(startup);

    // Restore the ACT
#   if ACT_SUPPORT
    OK = OK && ActStartup(startup);
#   endif

    // The following code was moved from Time.c where it made no sense
    if(OK)
    {
        switch(startup)
        {
            case SU_RESUME:
                // Resume sequence
                gr.restartCount++;
                break;
            case SU_RESTART:
                // Hibernate sequence
```

```
                gr.clearCount++;
                gr.restartCount++;
                break;
            case SU_RESET:
            default:
                // Reset object context ID to 0
                gr.objectContextID = 0;
                // Reset clearCount to 0
                gr.clearCount = 0;

                // Reset sequence
                // Increase resetCount
                gp.resetCount++;

                // Write resetCount to NV
                NV_SYNC_PERSISTENT(resetCount);

                gp.totalResetCount++;
                // We do not expect the total reset counter overflow during the life
                // time of TPM.  if it ever happens, TPM will be put to failure mode
                // and there is no way to recover it.
                // The reason that there is no recovery is that we don't increment
                // the NV totalResetCount when incrementing would make it 0. When the
                // TPM starts up again, the old value of totalResetCount will be read
                // and we will get right back to here with the increment failing.
                if(gp.totalResetCount == 0)
                    FAIL(FATAL_ERROR_INTERNAL);

                // Write total reset counter to NV
                NV_SYNC_PERSISTENT(totalResetCount);

                // Reset restartCount
                gr.restartCount = 0;

                break;
        }
    }
    // Initialize session table
    OK = OK && SessionStartup(startup);

    // Initialize object table
    OK = OK && ObjectStartup();

    // Initialize index/evict data.  This function clears read/write locks
    // in NV index
    OK = OK && NvEntityStartup(startup);

    // Initialize the orderly shut down flag for this cycle to SU_NONE_VALUE.
    gp.orderlyState = SU_NONE_VALUE;

    OK              = OK && NV_SYNC_PERSISTENT(orderlyState);

    // This can be reset after the first completion of a TPM2_Startup() after
    // a power loss. It can probably be reset earlier but this is an OK place.
    if(OK)
        g_powerWasLost = FALSE;

    return (OK) ? TPM_RC_SUCCESS : TPM_RC_FAILURE;
}

#endif  // CC_Startup
```

7.126 /tpm/src/command/Symmetric/EncryptDecrypt.c

```
#include "Tpm.h"
```

```
#include "EncryptDecrypt_fp.h"
#if CC_EncryptDecrypt2
#  include "EncryptDecrypt_spt_fp.h"
#endif

#if CC_EncryptDecrypt  // Conditional expansion of this file

/*(See part 3 specification)
// symmetric encryption or decryption
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          is not a symmetric decryption key with both
//                          public and private portions loaded
//      TPM_RC_SIZE         'IvIn' size is incompatible with the block cipher mode;
//                          or 'inData' size is not an even multiple of the block
//                          size for CBC or ECB mode
//      TPM_RC_VALUE        'keyHandle' is restricted and the argument 'mode' does
//                          not match the key's mode
TPM_RC
TPM2_EncryptDecrypt(EncryptDecrypt_In*  in,  // IN: input parameter list
                    EncryptDecrypt_Out* out  // OUT: output parameter list
)
{
#  if CC_EncryptDecrypt2
    return EncryptDecryptShared(
        in->keyHandle, in->decrypt, in->mode, &in->ivIn, &in->inData, out);
#  else
    OBJECT*     symKey;
    UINT16      keySize;
    UINT16      blockSize;
    BYTE*       key;
    TPM_ALG_ID  alg;
    TPM_ALG_ID  mode;
    TPM_RC      result;
    BOOL        OK;
    TPMA_OBJECT attributes;

    // Input Validation
    symKey     = HandleToObject(in->keyHandle);
    mode       = symKey->publicArea.parameters.symDetail.sym.mode.sym;
    attributes = symKey->publicArea.objectAttributes;

    // The input key should be a symmetric key
    if(symKey->publicArea.type != TPM_ALG_SYMCIPHER)
        return TPM_RCS_KEY + RC_EncryptDecrypt_keyHandle;
    // The key must be unrestricted and allow the selected operation
    OK      = IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted) if(YES == in->decrypt)
        OK  = OK && IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt);
    else OK = OK && IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign);
    if(!OK)
        return TPM_RCS_ATTRIBUTES + RC_EncryptDecrypt_keyHandle;

    // If the key mode is not TPM_ALG_NULL...
    // or TPM_ALG_NULL
    if(mode != TPM_ALG_NULL)
    {
        // then the input mode has to be TPM_ALG_NULL or the same as the key
        if((in->mode != TPM_ALG_NULL) && (in->mode != mode))
            return TPM_RCS_MODE + RC_EncryptDecrypt_mode;
    }
    else
    {
        // if the key mode is null, then the input can't be null
        if(in->mode == TPM_ALG_NULL)
            return TPM_RCS_MODE + RC_EncryptDecrypt_mode;
        mode = in->mode;
```

```
    }
    // The input iv for ECB mode should be an Empty Buffer.  All the other modes
    // should have an iv size same as encryption block size
    keySize   = symKey->publicArea.parameters.symDetail.sym.keyBits.sym;
    alg       = symKey->publicArea.parameters.symDetail.sym.algorithm;
    blockSize = CryptGetSymmetricBlockSize(alg, keySize);

    // reverify the algorithm. This is mainly to keep static analysis tools happy
    if(blockSize == 0)
        return TPM_RCS_KEY + RC_EncryptDecrypt_keyHandle;

    // Note: When an algorithm is not supported by a TPM, the TPM_ALG_xxx for that
    // algorithm is not defined. However, it is assumed that the TPM_ALG_xxx for
    // the algorithm is always defined. Both have the same numeric value.
    // TPM_ALG_xxx is used here so that the code does not get cluttered with
    // #ifdef's. Having this check does not mean that the algorithm is supported.
    // If it was not supported the unmarshaling code would have rejected it before
    // this function were called. This means that, depending on the implementation,
    // the check could be redundant but it doesn't hurt.
    if(((mode == TPM_ALG_ECB) && (in->ivIn.t.size != 0))
       || ((mode != TPM_ALG_ECB) && (in->ivIn.t.size != blockSize)))
        return TPM_RCS_SIZE + RC_EncryptDecrypt_ivIn;

    // The input data size of CBC mode or ECB mode must be an even multiple of
    // the symmetric algorithm's block size
    if(((mode == TPM_ALG_CBC) || (mode == TPM_ALG_ECB))
       && ((in->inData.t.size % blockSize) != 0))
        return TPM_RCS_SIZE + RC_EncryptDecrypt_inData;

    // Copy IV
    // Note: This is copied here so that the calls to the encrypt/decrypt functions
    // will modify the output buffer, not the input buffer
    out->ivOut = in->ivIn;

    // Command Output
    key = symKey->sensitive.sensitive.sym.t.buffer;
    // For symmetric encryption, the cipher data size is the same as plain data
    // size.
    out->outData.t.size = in->inData.t.size;
    if(in->decrypt == YES)
    {
        // Decrypt data to output
        result = CryptSymmetricDecrypt(out->outData.t.buffer,
                                       alg,
                                       keySize,
                                       key,
                                       &(out->ivOut),
                                       mode,
                                       in->inData.t.size,
                                       in->inData.t.buffer);
    }
    else
    {
        // Encrypt data to output
        result = CryptSymmetricEncrypt(out->outData.t.buffer,
                                       alg,
                                       keySize,
                                       key,
                                       &(out->ivOut),
                                       mode,
                                       in->inData.t.size,
                                       in->inData.t.buffer);
    }
    return result;
#  endif  // CC_EncryptDecrypt2
}
```

```
#endif  // CC_EncryptDecrypt
```

7.127 /tpm/src/command/Symmetric/EncryptDecrypt2.c

```
#include "Tpm.h"
#include "EncryptDecrypt2_fp.h"
#include "EncryptDecrypt_fp.h"
#include "EncryptDecrypt_spt_fp.h"

#if CC_EncryptDecrypt2  // Conditional expansion of this file

/*(See part 3 specification)
// symmetric encryption or decryption using modified parameter list
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          is not a symmetric decryption key with both
//                          public and private portions loaded
//      TPM_RC_SIZE         'IvIn' size is incompatible with the block cipher mode;
//                          or 'inData' size is not an even multiple of the block
//                          size for CBC or ECB mode
//      TPM_RC_VALUE        'keyHandle' is restricted and the argument 'mode' does
//                          not match the key's mode
TPM_RC
TPM2_EncryptDecrypt2(EncryptDecrypt2_In*  in,  // IN: input parameter list
                     EncryptDecrypt2_Out* out  // OUT: output parameter list
)
{
    TPM_RC result;
    // EncryptDecyrptShared() performs the operations as shown in
    // TPM2_EncrypDecrypt
    result = EncryptDecryptShared(in->keyHandle,
                                  in->decrypt,
                                  in->mode,
                                  &in->ivIn,
                                  &in->inData,
                                  (EncryptDecrypt_Out*)out);
    // Handle response code swizzle.
    switch(result)
    {
        case TPM_RCS_MODE + RC_EncryptDecrypt_mode:
            result = TPM_RCS_MODE + RC_EncryptDecrypt2_mode;
            break;
        case TPM_RCS_SIZE + RC_EncryptDecrypt_ivIn:
            result = TPM_RCS_SIZE + RC_EncryptDecrypt2_ivIn;
            break;
        case TPM_RCS_SIZE + RC_EncryptDecrypt_inData:
            result = TPM_RCS_SIZE + RC_EncryptDecrypt2_inData;
            break;
        default:
            break;
    }
    return result;
}

#endif  // CC_EncryptDecrypt2
```

7.128 /tpm/src/command/Symmetric/EncryptDecrypt_spt.c

```
#include "Tpm.h"
#include "EncryptDecrypt_fp.h"
#include "EncryptDecrypt_spt_fp.h"

#if CC_EncryptDecrypt2
```

```
/*(See part 3 specification)
// symmetric encryption or decryption
*/
//  Return Type: TPM_RC
//      TPM_RC_KEY          is not a symmetric decryption key with both
//                          public and private portions loaded
//      TPM_RC_SIZE         'IvIn' size is incompatible with the block cipher mode;
//                          or 'inData' size is not an even multiple of the block
//                          size for CBC or ECB mode
//      TPM_RC_VALUE        'keyHandle' is restricted and the argument 'mode' does
//                          not match the key's mode
TPM_RC
EncryptDecryptShared(TPMI_DH_OBJECT      keyHandleIn,
                     TPMI_YES_NO         decryptIn,
                     TPMI_ALG_SYM_MODE   modeIn,
                     TPM2B_IV*           ivIn,
                     TPM2B_MAX_BUFFER*   inData,
                     EncryptDecrypt_Out* out)
{
    OBJECT*     symKey;
    UINT16      keySize;
    UINT16      blockSize;
    BYTE*       key;
    TPM_ALG_ID alg;
    TPM_ALG_ID mode;
    TPM_RC      result;
    BOOL        OK;
    // Input Validation
    symKey = HandleToObject(keyHandleIn);
    mode   = symKey->publicArea.parameters.symDetail.sym.mode.sym;

    // The input key should be a symmetric key
    if(symKey->publicArea.type != TPM_ALG_SYMCIPHER)
        return TPM_RCS_KEY + RC_EncryptDecrypt_keyHandle;
    // The key must be unrestricted and allow the selected operation
    OK = !IS_ATTRIBUTE(symKey->publicArea.objectAttributes, TPMA_OBJECT, restricted);
    if(YES == decryptIn)
        OK = OK
             && IS_ATTRIBUTE(
                 symKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt);
    else
        OK = OK
             && IS_ATTRIBUTE(symKey->publicArea.objectAttributes, TPMA_OBJECT, sign);
    if(!OK)
        return TPM_RCS_ATTRIBUTES + RC_EncryptDecrypt_keyHandle;

    // Make sure that key is an encrypt/decrypt key and not SMAC
    if(!CryptSymModeIsValid(mode, TRUE))
        return TPM_RCS_MODE + RC_EncryptDecrypt_keyHandle;

    // If the key mode is not TPM_ALG_NULL...
    // or TPM_ALG_NULL
    if(mode != TPM_ALG_NULL)
    {
        // then the input mode has to be TPM_ALG_NULL or the same as the key
        if((modeIn != TPM_ALG_NULL) && (modeIn != mode))
            return TPM_RCS_MODE + RC_EncryptDecrypt_mode;
    }
    else
    {
        // if the key mode is null, then the input can't be null
        if(modeIn == TPM_ALG_NULL)
            return TPM_RCS_MODE + RC_EncryptDecrypt_mode;
        mode = modeIn;
    }
```

```
    // The input iv for ECB mode should be an Empty Buffer.  All the other modes
    // should have an iv size same as encryption block size
    keySize   = symKey->publicArea.parameters.symDetail.sym.keyBits.sym;
    alg       = symKey->publicArea.parameters.symDetail.sym.algorithm;
    blockSize = CryptGetSymmetricBlockSize(alg, keySize);

    // reverify the algorithm. This is mainly to keep static analysis tools happy
    if(blockSize == 0)
        return TPM_RCS_KEY + RC_EncryptDecrypt_keyHandle;

    if(((mode == TPM_ALG_ECB) && (ivIn->t.size != 0))
       || ((mode != TPM_ALG_ECB) && (ivIn->t.size != blockSize)))
        return TPM_RCS_SIZE + RC_EncryptDecrypt_ivIn;

    // The input data size of CBC mode or ECB mode must be an even multiple of
    // the symmetric algorithm's block size
    if(((mode == TPM_ALG_CBC) || (mode == TPM_ALG_ECB))
       && ((inData->t.size % blockSize) != 0))
        return TPM_RCS_SIZE + RC_EncryptDecrypt_inData;

    // Copy IV
    // Note: This is copied here so that the calls to the encrypt/decrypt functions
    // will modify the output buffer, not the input buffer
    out->ivOut = *ivIn;

    // Command Output
    key = symKey->sensitive.sensitive.sym.t.buffer;
    // For symmetric encryption, the cipher data size is the same as plain data
    // size.
    out->outData.t.size = inData->t.size;
    if(decryptIn == YES)
    {
        // Decrypt data to output
        result = CryptSymmetricDecrypt(out->outData.t.buffer,
                                       alg,
                                       keySize,
                                       key,
                                       &(out->ivOut),
                                       mode,
                                       inData->t.size,
                                       inData->t.buffer);
    }
    else
    {
        // Encrypt data to output
        result = CryptSymmetricEncrypt(out->outData.t.buffer,
                                       alg,
                                       keySize,
                                       key,
                                       &(out->ivOut),
                                       mode,
                                       inData->t.size,
                                       inData->t.buffer);
    }
    return result;
}

#endif  // CC_EncryptDecrypt
```

7.129 /tpm/src/command/Symmetric/Hash.c

```
#include "Tpm.h"
#include "Hash_fp.h"

#if CC_Hash  // Conditional expansion of this file
```

```
/*(See part 3 specification)
// Hash a data buffer
*/
TPM_RC
TPM2_Hash(Hash_In*  in,  // IN: input parameter list
          Hash_Out* out  // OUT: output parameter list
)
{
    HASH_STATE hashState;

    // Command Output

    // Output hash
    // Start hash stack
    out->outHash.t.size = CryptHashStart(&hashState, in->hashAlg);
    // Adding hash data
    CryptDigestUpdate2B(&hashState, &in->data.b);
    // Complete hash
    CryptHashEnd2B(&hashState, &out->outHash.b);

    // Output ticket
    out->validation.tag       = TPM_ST_HASHCHECK;
    out->validation.hierarchy = in->hierarchy;

    if(in->hierarchy == TPM_RH_NULL)
    {
        // Ticket is not required
        out->validation.hierarchy     = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else if(
        in->data.t.size >= sizeof(TPM_GENERATED_VALUE) && !TicketIsSafe(&in->data.b))
    {
        // Ticket is not safe
        out->validation.hierarchy     = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else
    {
        TPM_RC result;
        // Compute ticket
        result = TicketComputeHashCheck(
            in->hierarchy, in->hashAlg, &out->outHash, &out->validation);
        if(result != TPM_RC_SUCCESS)
            return result;
    }

    return TPM_RC_SUCCESS;
}

#endif  // CC_Hash
```

7.130 /tpm/src/command/Symmetric/HMAC.c

```
#include "Tpm.h"
#include "HMAC_fp.h"

#if CC_HMAC  // Conditional expansion of this file

/*(See part 3 specification)
// Compute HMAC on a data buffer
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       key referenced by 'handle' is a restricted key
```

```
//      TPM_RC_KEY              'handle' does not reference a signing key
//      TPM_RC_TYPE             key referenced by 'handle' is not an HMAC key
//      TPM_RC_VALUE           'hashAlg' is not compatible with the hash algorithm
//                              of the scheme of the object referenced by 'handle'
TPM_RC
TPM2_HMAC(HMAC_In*  in,  // IN: input parameter list
          HMAC_Out* out  // OUT: output parameter list
)
{
    HMAC_STATE    hmacState;
    OBJECT*       hmacObject;
    TPMI_ALG_HASH hashAlg;
    TPMT_PUBLIC*  publicArea;

    // Input Validation

    // Get HMAC key object and public area pointers
    hmacObject = HandleToObject(in->handle);
    publicArea = &hmacObject->publicArea;
    // Make sure that the key is an HMAC key
    if(publicArea->type != TPM_ALG_KEYEDHASH)
        return TPM_RCS_TYPE + RC_HMAC_handle;

    // and that it is unrestricted
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted))
        return TPM_RCS_ATTRIBUTES + RC_HMAC_handle;

    // and that it is a signing key
    if(!IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
        return TPM_RCS_KEY + RC_HMAC_handle;

    // See if the key has a default
    if(publicArea->parameters.keyedHashDetail.scheme.scheme == TPM_ALG_NULL)
        // it doesn't so use the input value
        hashAlg = in->hashAlg;
    else
    {
        // key has a default so use it
        hashAlg = publicArea->parameters.keyedHashDetail.scheme.details.hmac.hashAlg;
        // and verify that the input was either the  TPM_ALG_NULL or the default
        if(in->hashAlg != TPM_ALG_NULL && in->hashAlg != hashAlg)
            hashAlg = TPM_ALG_NULL;
    }
    // if we ended up without a hash algorithm then return an error
    if(hashAlg == TPM_ALG_NULL)
        return TPM_RCS_VALUE + RC_HMAC_hashAlg;

    // Command Output

    // Start HMAC stack
    out->outHMAC.t.size = CryptHmacStart2B(
        &hmacState, hashAlg, &hmacObject->sensitive.sensitive.bits.b);
    // Adding HMAC data
    CryptDigestUpdate2B(&hmacState.hashState, &in->buffer.b);

    // Complete HMAC
    CryptHmacEnd2B(&hmacState, &out->outHMAC.b);

    return TPM_RC_SUCCESS;
}

#endif  // CC_HMAC
```

7.131 /tpm/src/command/Symmetric/MAC.c

```
#include "Tpm.h"
#include "MAC_fp.h"

#if CC_MAC  // Conditional expansion of this file

/*(See part 3 specification)
// Compute MAC on a data buffer
*/
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES       key referenced by 'handle' is a restricted key
//      TPM_RC_KEY              'handle' does not reference a signing key
//      TPM_RC_TYPE             key referenced by 'handle' is not an HMAC key
//      TPM_RC_VALUE           'hashAlg' is not compatible with the hash algorithm
//                              of the scheme of the object referenced by 'handle'
TPM_RC
TPM2_MAC(MAC_In*  in,  // IN: input parameter list
         MAC_Out* out  // OUT: output parameter list
)
{
    OBJECT*      keyObject;
    HMAC_STATE   state;
    TPMT_PUBLIC* publicArea;
    TPM_RC       result;

    // Input Validation
    // Get MAC key object and public area pointers
    keyObject  = HandleToObject(in->handle);
    publicArea = &keyObject->publicArea;

    // If the key is not able to do a MAC, indicate that the handle selects an
    // object that can't do a MAC
    result = CryptSelectMac(publicArea, &in->inScheme);
    if(result == TPM_RCS_TYPE)
        return TPM_RCS_TYPE + RC_MAC_handle;
    // If there is another error type, indicate that the scheme and key are not
    // compatible
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_MAC_inScheme);
    // Make sure that the key is not restricted
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted))
        return TPM_RCS_ATTRIBUTES + RC_MAC_handle;
    // and that it is a signing key
    if(!IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
        return TPM_RCS_KEY + RC_MAC_handle;
    // Command Output
    out->outMAC.t.size = CryptMacStart(&state,
                                       &publicArea->parameters,
                                       in->inScheme,
                                       &keyObject->sensitive.sensitive.any.b);
    // If the mac can't start, treat it as a fatal error
    if(out->outMAC.t.size == 0)
        return TPM_RC_FAILURE;
    CryptDigestUpdate2B(&state.hashState, &in->buffer.b);
    // If the MAC result is not what was expected, it is a fatal error
    if(CryptHmacEnd2B(&state, &out->outMAC.b) != out->outMAC.t.size)
        return TPM_RC_FAILURE;
    return TPM_RC_SUCCESS;
}

#endif  // CC_MAC
```

7.132 /tpm/src/command/Testing/GetTestResult.c

```
#include "Tpm.h"
#include "GetTestResult_fp.h"

#if CC_GetTestResult  // Conditional expansion of this file

/*(See part 3 specification)
// returns manufacturer-specific information regarding the results of a self-
// test and an indication of the test status.
*/

// In the reference implementation, this function is only reachable if the TPM is
// not in failure mode meaning that all tests that have been run have completed
// successfully. There is not test data and the test result is TPM_RC_SUCCESS.
TPM_RC
TPM2_GetTestResult(GetTestResult_Out* out  // OUT: output parameter list
)
{
    // Command Output

    // Call incremental self test function in crypt module
    out->testResult = CryptGetTestResult(&out->outData);

    return TPM_RC_SUCCESS;
}

#endif  // CC_GetTestResult
```

7.133 /tpm/src/command/Testing/IncrementalSelfTest.c

```
#include "Tpm.h"
#include "IncrementalSelfTest_fp.h"

#if CC_IncrementalSelfTest  // Conditional expansion of this file

/*(See part 3 specification)
// perform a test of selected algorithms
*/
//  Return Type: TPM_RC
//      TPM_RC_CANCELED         the command was canceled (some tests may have
//                              completed)
//      TPM_RC_VALUE            an algorithm in the toTest list is not implemented
TPM_RC
TPM2_IncrementalSelfTest(IncrementalSelfTest_In*  in,  // IN: input parameter list
                         IncrementalSelfTest_Out* out  // OUT: output parameter list
)
{
    TPM_RC result;
    // Command Output

    // Call incremental self test function in crypt module. If this function
    // returns TPM_RC_VALUE, it means that an algorithm on the 'toTest' list is
    // not implemented.
    result = CryptIncrementalSelfTest(&in->toTest, &out->toDoList);
    if(result == TPM_RC_VALUE)
        return TPM_RCS_VALUE + RC_IncrementalSelfTest_toTest;
    return result;
}

#endif  // CC_IncrementalSelfTest
```

7.134 /tpm/src/command/Testing/SelfTest.c

```
#include "Tpm.h"
#include "SelfTest_fp.h"

#if CC_SelfTest  // Conditional expansion of this file

/*(See part 3 specification)
// perform a test of TPM capabilities
*/
//  Return Type: TPM_RC
//      TPM_RC_CANCELED             the command was canceled (some incremental
//                                  process may have been made)
//      TPM_RC_TESTING              self test in process
TPM_RC
TPM2_SelfTest(SelfTest_In* in  // IN: input parameter list
)
{
    // Command Output

    // Call self test function in crypt module
    return CryptSelfTest(in->fullTest);
}

#endif  // CC_SelfTest
```

7.135 /tpm/src/command/Vendor/Vendor_TCG_Test.c

```
#include "Tpm.h"

#if CC_Vendor_TCG_Test  // Conditional expansion of this file
#  include "Vendor_TCG_Test_fp.h"

TPM_RC
TPM2_Vendor_TCG_Test(Vendor_TCG_Test_In*  in,  // IN: input parameter list
                     Vendor_TCG_Test_Out* out  // OUT: output parameter list
)
{
    out->outputData = in->inputData;
    return TPM_RC_SUCCESS;
}

#endif  // CC_Vendor_TCG_Test
```

7.136 /tpm/src/crypt/AlgorithmTests.c

```
//** Introduction
// This file contains the code to perform the various self-test functions.
//
// NOTE: In this implementation, large local variables are made static to minimize
// stack usage, which is critical for stack-constrained platforms.

//** Includes and Defines
#include "Tpm.h"

#define SELF_TEST_DATA

#if ENABLE_SELF_TESTS

// These includes pull in the data structures. They contain data definitions for the
// various tests.
#  include "SelfTest.h"
#  include "SymmetricTest.h"
```

```
#  include "RsaTestData.h"
#  include "EccTestData.h"
#  include "HashTestData.h"
#  include "KdfTestData.h"

#  define TEST_DEFAULT_TEST_HASH(vector)        \
      if(TEST_BIT(DEFAULT_TEST_HASH, g_toTest)) \
          TestHash(DEFAULT_TEST_HASH, vector);

// Make sure that the algorithm has been tested
#  define CLEAR_BOTH(alg)                \
      {                                  \
          CLEAR_BIT(alg, *toTest);       \
          if(toTest != &g_toTest)        \
              CLEAR_BIT(alg, g_toTest); \
      }

#  define SET_BOTH(alg)                \
      {                                \
          SET_BIT(alg, *toTest);       \
          if(toTest != &g_toTest)      \
              SET_BIT(alg, g_toTest); \
      }

#  define TEST_BOTH(alg)                                                          \
      ((toTest != &g_toTest) ? TEST_BIT(alg, *toTest) || TEST_BIT(alg, g_toTest) \
                             : TEST_BIT(alg, *toTest))

// Can only cancel if doing a list.
#  define CHECK_CANCELED                                \
      if(_plat__IsCanceled() && toTest != &g_toTest) \
          return TPM_RC_CANCELED;

//** Hash Tests

//*** Description
// The hash test does a known-value HMAC using the specified hash algorithm.

//*** TestHash()
// The hash test function.
static TPM_RC TestHash(TPM_ALG_ID hashAlg, ALGORITHM_VECTOR* toTest)
{
    static TPM2B_DIGEST computed;  // value computed
    static HMAC_STATE   state;
    UINT16              digestSize;
    const TPM2B*        testDigest = NULL;
    //    TPM2B_TYPE(HMAC_BLOCK, DEFAULT_TEST_HASH_BLOCK_SIZE);

    pAssert(hashAlg != TPM_ALG_NULL);
#  define HASH_CASE_FOR_TEST(HASH, hash)     \
      case ALG_##HASH##_VALUE:               \
          testDigest = &c_##HASH##_digest.b; \
          break;
    switch(hashAlg)
    {
        FOR_EACH_HASH(HASH_CASE_FOR_TEST)

        default:
            FAIL(FATAL_ERROR_INTERNAL);
    }
    // Clear the to-test bits
    CLEAR_BOTH(hashAlg);

    // If there is an algorithm without test vectors, then assume that things are OK.
    if(testDigest == NULL || testDigest->size == 0)
        return TPM_RC_SUCCESS;
```

```
    // Set the HMAC key to twice the digest size
    digestSize = CryptHashGetDigestSize(hashAlg);
    CryptHmacStart(&state, hashAlg, digestSize * 2, (BYTE*)c_hashTestKey.t.buffer);
    CryptDigestUpdate(&state.hashState,
                      2 * CryptHashGetBlockSize(hashAlg),
                      (BYTE*)c_hashTestData.t.buffer);
    computed.t.size = digestSize;
    CryptHmacEnd(&state, digestSize, computed.t.buffer);
    if((testDigest->size != computed.t.size)
       || (memcmp(testDigest->buffer, computed.t.buffer, computed.b.size) != 0))
        SELF_TEST_FAILURE;
    return TPM_RC_SUCCESS;
}

//** Symmetric Test Functions

//*** MakeIv()
// Internal function to make the appropriate IV depending on the mode.
static UINT32 MakeIv(TPM_ALG_ID mode,  // IN: symmetric mode
                     UINT32     size,  // IN: block size of the algorithm
                     BYTE*      iv     // OUT: IV to fill in
)
{
    BYTE i;

    if(mode == TPM_ALG_ECB)
        return 0;
    if(mode == TPM_ALG_CTR)
    {
        // The test uses an IV that has 0xff in the last byte
        for(i = 1; i <= size; i++)
            *iv++ = 0xff - (BYTE)(size - i);
    }
    else
    {
        for(i = 0; i < size; i++)
            *iv++ = i;
    }
    return size;
}

//*** TestSymmetricAlgorithm()
// Function to test a specific algorithm, key size, and mode.
static void TestSymmetricAlgorithm(const SYMMETRIC_TEST_VECTOR* test,  //
                                   TPM_ALG_ID                   mode   //
)
{
    static BYTE     encrypted[MAX_SYM_BLOCK_SIZE * 2];
    static BYTE     decrypted[MAX_SYM_BLOCK_SIZE * 2];
    static TPM2B_IV iv;
    //
    // Get the appropriate IV
    iv.t.size = (UINT16)MakeIv(mode, test->ivSize, iv.t.buffer);

    // Encrypt known data
    CryptSymmetricEncrypt(encrypted,
                          test->alg,
                          test->keyBits,
                          test->key,
                          &iv,
                          mode,
                          test->dataInOutSize,
                          test->dataIn);
    // Check that it matches the expected value
    if(!MemoryEqual(
```

```
            encrypted, test->dataOut[mode - TPM_ALG_CTR], test->dataInOutSize))
        SELF_TEST_FAILURE;
    // Reinitialize the iv for decryption
    MakeIv(mode, test->ivSize, iv.t.buffer);
    CryptSymmetricDecrypt(decrypted,
                          test->alg,
                          test->keyBits,
                          test->key,
                          &iv,
                          mode,
                          test->dataInOutSize,
                          test->dataOut[mode - TPM_ALG_CTR]);
    // Make sure that it matches what we started with
    if(!MemoryEqual(decrypted, test->dataIn, test->dataInOutSize))
        SELF_TEST_FAILURE;
}

//*** AllSymsAreDone()
// Checks if both symmetric algorithms have been tested. This is put here
// so that addition of a symmetric algorithm will be relatively easy to handle.
//
//  Return Type: BOOL
//      TRUE(1)         all symmetric algorithms tested
//      FALSE(0)        not all symmetric algorithms tested
static BOOL AllSymsAreDone(ALGORITHM_VECTOR* toTest)
{
    return (!TEST_BOTH(TPM_ALG_AES) && !TEST_BOTH(TPM_ALG_SM4));
}

//*** AllModesAreDone()
// Checks if all the modes have been tested.
//
//  Return Type: BOOL
//      TRUE(1)         all modes tested
//      FALSE(0)        all modes not tested
static BOOL AllModesAreDone(ALGORITHM_VECTOR* toTest)
{
    TPM_ALG_ID alg;
    for(alg = SYM_MODE_FIRST; alg <= SYM_MODE_LAST; alg++)
        if(TEST_BOTH(alg))
            return FALSE;
    return TRUE;
}

//*** TestSymmetric()
// If 'alg' is a symmetric block cipher, then all of the modes that are selected are
// tested. If 'alg' is a mode, then all algorithms of that mode are tested.
static TPM_RC TestSymmetric(TPM_ALG_ID alg, ALGORITHM_VECTOR* toTest)
{
    SYM_INDEX  index;
    TPM_ALG_ID mode;
    //
    if(!TEST_BIT(alg, *toTest))
        return TPM_RC_SUCCESS;
    if(alg == TPM_ALG_AES || alg == TPM_ALG_SM4 || alg == TPM_ALG_CAMELLIA)
    {
        // Will test the algorithm for all modes and key sizes
        CLEAR_BOTH(alg);

        // A test this algorithm for all modes
        for(index = 0; index < NUM_SYMS; index++)
        {
            if(c_symTestValues[index].alg == alg)
            {
                for(mode = SYM_MODE_FIRST; mode <= SYM_MODE_LAST; mode++)
                {
```

```
                    if(TEST_BIT(mode, *toTest))
                        TestSymmetricAlgorithm(&c_symTestValues[index], mode);
                }
            }
        }
        // if all the symmetric tests are done
        if(AllSymsAreDone(toTest))
        {
            // all symmetric algorithms tested so no modes should be set
            for(alg = SYM_MODE_FIRST; alg <= SYM_MODE_LAST; alg++)
                CLEAR_BOTH(alg);
        }
    }
    else if(SYM_MODE_FIRST <= alg && alg <= SYM_MODE_LAST)
    {
        // Test this mode for all key sizes and algorithms
        for(index = 0; index < NUM_SYMS; index++)
        {
            // The mode testing only comes into play when doing self tests
            // by command. When doing self tests by command, the block ciphers are
            // tested first. That means that all of their modes would have been
            // tested for all key sizes. If there is no block cipher left to
            // test, then clear this mode bit.
            if(!TEST_BIT(TPM_ALG_AES, *toTest) && !TEST_BIT(TPM_ALG_SM4, *toTest))
            {
                CLEAR_BOTH(alg);
            }
            else
            {
                for(index = 0; index < NUM_SYMS; index++)
                {
                    if(TEST_BIT(c_symTestValues[index].alg, *toTest))
                        TestSymmetricAlgorithm(&c_symTestValues[index], alg);
                }
                // have tested this mode for all algorithms
                CLEAR_BOTH(alg);
            }
        }
        if(AllModesAreDone(toTest))
        {
            CLEAR_BOTH(TPM_ALG_AES);
            CLEAR_BOTH(TPM_ALG_SM4);
        }
    }
    else
        pAssert(alg == 0 && alg != 0);
    return TPM_RC_SUCCESS;
}

//** RSA Tests
#  if ALG_RSA

//*** Introduction
// The tests are for public key only operations and for private key operations.
// Signature verification and encryption are public key operations. They are tested
// by using a KVT. For signature verification, this means that a known good
// signature is checked by CryptRsaValidateSignature(). If it fails, then the
// TPM enters failure mode. For encryption, the TPM encrypts known values using
// the selected scheme and checks that the returned value matches the expected
// value.
//
// For private key operations, a full scheme check is used. For a signing key, a
// known key is used to sign a known message. Then that signature is verified.
// since the signature may involve use of random values, the signature will be
// different each time and we can't always check that the signature matches a
// known value. The same technique is used for decryption (RSADP/RSAEP).
```

```
//
// When an operation uses the public key and the verification has not been
// tested, the TPM will do a KVT.
//
// The test for the signing algorithm is built into the call for the algorithm

//*** RsaKeyInitialize()
// The test key is defined by a public modulus and a private prime. The TPM's RSA
// code computes the second prime and the private exponent.
static void RsaKeyInitialize(OBJECT* testObject)
{
    MemoryCopy2B(&testObject->publicArea.unique.rsa.b,
                 (P2B)&c_rsaPublicModulus,
                 sizeof(c_rsaPublicModulus));
    MemoryCopy2B(&testObject->sensitive.sensitive.rsa.b,
                 (P2B)&c_rsaPrivatePrime,
                 sizeof(testObject->sensitive.sensitive.rsa.t.buffer));
    testObject->publicArea.parameters.rsaDetail.keyBits = RSA_TEST_KEY_SIZE * 8;
    // Use the default exponent
    testObject->publicArea.parameters.rsaDetail.exponent = 0;
}

//*** TestRsaEncryptDecrypt()
// These tests are for a public key encryption that uses a random value.
static TPM_RC TestRsaEncryptDecrypt(TPM_ALG_ID        scheme,  // IN: the scheme
                                    ALGORITHM_VECTOR* toTest   //
)
{
    static TPM2B_PUBLIC_KEY_RSA testInput;
    static TPM2B_PUBLIC_KEY_RSA testOutput;
    static OBJECT               testObject;
    const TPM2B_RSA_TEST_KEY*   kvtValue  = NULL;
    TPM_RC                      result    = TPM_RC_SUCCESS;
    const TPM2B*                testLabel = NULL;
    TPMT_RSA_DECRYPT            rsaScheme;
    //
    // Don't need to initialize much of the test object
    RsaKeyInitialize(&testObject);
    rsaScheme.scheme                  = scheme;
    rsaScheme.details.anySig.hashAlg = DEFAULT_TEST_HASH;
    CLEAR_BOTH(scheme);
    CLEAR_BOTH(TPM_ALG_NULL);
    if(scheme == TPM_ALG_NULL)
    {
        // This is an encryption scheme using the private key without any encoding.
        memcpy(testInput.t.buffer, c_RsaTestValue, sizeof(c_RsaTestValue));
        testInput.t.size = sizeof(c_RsaTestValue);
        if(TPM_RC_SUCCESS
           != CryptRsaEncrypt(
               &testOutput, &testInput.b, &testObject, &rsaScheme, NULL, NULL))
            SELF_TEST_FAILURE;
        if(!MemoryEqual(testOutput.t.buffer, c_RsaepKvt.buffer, c_RsaepKvt.size))
            SELF_TEST_FAILURE;
        MemoryCopy2B(&testInput.b, &testOutput.b, sizeof(testInput.t.buffer));
        if(TPM_RC_SUCCESS
           != CryptRsaDecrypt(
               &testOutput.b, &testInput.b, &testObject, &rsaScheme, NULL))
            SELF_TEST_FAILURE;
        if(!MemoryEqual(testOutput.t.buffer, c_RsaTestValue, sizeof(c_RsaTestValue)))
            SELF_TEST_FAILURE;
    }
    else
    {
        // TPM_ALG_RSAES:
        // This is an decryption scheme using padding according to
        // PKCS#1v2.1, 7.2. This padding uses random bits. To test a public
```

```
        // key encryption that uses random data, encrypt a value and then
        // decrypt the value and see that we get the encrypted data back.
        // The hash is not used by this encryption so it can be TMP_ALG_NULL

        // TPM_ALG_OAEP:
        // This is also an decryption scheme and it also uses a
        // pseudo-random
        // value. However, this also uses a hash algorithm. So, we may need
        // to test that algorithm before use.
        if(scheme == TPM_ALG_OAEP)
        {
            TEST_DEFAULT_TEST_HASH(toTest);
            kvtValue  = &c_OaepKvt;
            testLabel = OAEP_TEST_STRING;
        }
        else if(scheme == TPM_ALG_RSAES)
        {
            kvtValue  = &c_RsaesKvt;
            testLabel = NULL;
        }
        else
            SELF_TEST_FAILURE;
        // Only use a digest-size portion of the test value
        memcpy(testInput.t.buffer, c_RsaTestValue, DEFAULT_TEST_DIGEST_SIZE);
        testInput.t.size = DEFAULT_TEST_DIGEST_SIZE;

        // See if the encryption works
        if(TPM_RC_SUCCESS
           != CryptRsaEncrypt(
               &testOutput, &testInput.b, &testObject, &rsaScheme, testLabel, NULL))
            SELF_TEST_FAILURE;
        MemoryCopy2B(&testInput.b, &testOutput.b, sizeof(testInput.t.buffer));
        // see if we can decrypt this value and get the original data back
        if(TPM_RC_SUCCESS
           != CryptRsaDecrypt(
               &testOutput.b, &testInput.b, &testObject, &rsaScheme, testLabel))
            SELF_TEST_FAILURE;
        // See if the results compare
        if(testOutput.t.size != DEFAULT_TEST_DIGEST_SIZE
           || !MemoryEqual(
               testOutput.t.buffer, c_RsaTestValue, DEFAULT_TEST_DIGEST_SIZE))
            SELF_TEST_FAILURE;
        // Now check that the decryption works on a known value
        MemoryCopy2B(&testInput.b, (P2B)kvtValue, sizeof(testInput.t.buffer));
        if(TPM_RC_SUCCESS
           != CryptRsaDecrypt(
               &testOutput.b, &testInput.b, &testObject, &rsaScheme, testLabel))
            SELF_TEST_FAILURE;
        if(testOutput.t.size != DEFAULT_TEST_DIGEST_SIZE
           || !MemoryEqual(
               testOutput.t.buffer, c_RsaTestValue, DEFAULT_TEST_DIGEST_SIZE))
            SELF_TEST_FAILURE;
    }
    return result;
}

//*** TestRsaSignAndVerify()
// This function does the testing of the RSA sign and verification functions. This
// test does a KVT.
static TPM_RC TestRsaSignAndVerify(TPM_ALG_ID scheme, ALGORITHM_VECTOR* toTest)
{
    TPM_RC                result = TPM_RC_SUCCESS;
    static OBJECT         testObject;
    static TPM2B_DIGEST   testDigest;
    static TPMT_SIGNATURE testSig;
```

```
    // Do a sign and signature verification.
    // RSASSA:
    // This is a signing scheme according to PKCS#1-v2.1 8.2. It does not
    // use random data so there is a KVT for the signing operation. On
    // first use of the scheme for signing, use the TPM's RSA key to
    // sign a portion of c_RsaTestData and compare the results to c_RsassaKvt. Then
    // decrypt the data to see that it matches the starting value. This verifies
    // the signature with a KVT

    // Clear the bits indicating that the function has not been checked. This is to
    // prevent looping
    CLEAR_BOTH(scheme);
    CLEAR_BOTH(TPM_ALG_NULL);
    CLEAR_BOTH(TPM_ALG_RSA);

    RsaKeyInitialize(&testObject);
    memcpy(testDigest.t.buffer, (BYTE*)c_RsaTestValue, DEFAULT_TEST_DIGEST_SIZE);
    testDigest.t.size             = DEFAULT_TEST_DIGEST_SIZE;
    testSig.sigAlg                = scheme;
    testSig.signature.rsapss.hash = DEFAULT_TEST_HASH;

    // RSAPSS:
    // This is a signing scheme a according to PKCS#1-v2.2 8.1 it uses
    // random data in the signature so there is no KVT for the signing
    // operation. To test signing, the TPM will use the TPM's RSA key
    // to sign a portion of c_RsaTestValue and then it will verify the
    // signature. For verification, c_RsapssKvt is verified before the
    // user signature blob is verified. The worst case for testing of this
    // algorithm is two private and one public key operation.

    // The process is to sign known data. If RSASSA is being done, verify that the
    // signature matches the precomputed value. For both, use the signed value and
    // see that the verification says that it is a good signature. Then
    // if testing RSAPSS, do a verify of a known good signature. This ensures that
    // the validation function works.

    if(TPM_RC_SUCCESS != CryptRsaSign(&testSig, &testObject, &testDigest, NULL))
        SELF_TEST_FAILURE;
    // For RSASSA, make sure the results is what we are looking for
    if(testSig.sigAlg == TPM_ALG_RSASSA)
    {
        if(testSig.signature.rsassa.sig.t.size != RSA_TEST_KEY_SIZE
           || !MemoryEqual(c_RsassaKvt.buffer,
                           testSig.signature.rsassa.sig.t.buffer,
                           RSA_TEST_KEY_SIZE))
            SELF_TEST_FAILURE;
    }
    // See if the TPM will validate its own signatures
    if(TPM_RC_SUCCESS
       != CryptRsaValidateSignature(&testSig, &testObject, &testDigest))
        SELF_TEST_FAILURE;
    // If this is RSAPSS, check the verification with known signature
    // Have to copy because  CrytpRsaValidateSignature() eats the signature
    if(TPM_ALG_RSAPSS == scheme)
    {
        MemoryCopy2B(&testSig.signature.rsapss.sig.b,
                     (P2B)&c_RsapssKvt,
                     sizeof(testSig.signature.rsapss.sig.t.buffer));
        if(TPM_RC_SUCCESS
           != CryptRsaValidateSignature(&testSig, &testObject, &testDigest))
            SELF_TEST_FAILURE;
    }
    return result;
}

//*** TestRSA()
```

```
// Function uses the provided vector to indicate which tests to run. It will clear
// the vector after each test is run and also clear g_toTest
static TPM_RC TestRsa(TPM_ALG_ID alg, ALGORITHM_VECTOR* toTest)
{
    TPM_RC result = TPM_RC_SUCCESS;
    //
    switch(alg)
    {
        case TPM_ALG_NULL:
            // This is the RSAEP/RSADP function. If we are processing a list, don't
            // need to test these now because any other test will validate
            // RSAEP/RSADP. Can tell this is list of test by checking to see if
            // 'toTest' is pointing at g_toTest. If so, this is an isolated test
            // an need to go ahead and do the test;
            if((toTest == &g_toTest)
               || (!TEST_BIT(TPM_ALG_RSASSA, *toTest)
                   && !TEST_BIT(TPM_ALG_RSAES, *toTest)
                   && !TEST_BIT(TPM_ALG_RSAPSS, *toTest)
                   && !TEST_BIT(TPM_ALG_OAEP, *toTest)))
                // Not running a list of tests or no other tests on the list
                // so run the test now
                result = TestRsaEncryptDecrypt(alg, toTest);
            // if not running the test now, leave the bit on, just in case things
            // get interrupted
            break;
        case TPM_ALG_OAEP:
        case TPM_ALG_RSAES:
            result = TestRsaEncryptDecrypt(alg, toTest);
            break;
        case TPM_ALG_RSAPSS:
        case TPM_ALG_RSASSA:
            result = TestRsaSignAndVerify(alg, toTest);
            break;
        default:
            SELF_TEST_FAILURE;
    }
    return result;
}

#  endif  // ALG_RSA

//** ECC Tests

#  if ALG_ECC

//*** LoadEccParameter()
// This function is mostly for readability and type checking
static void LoadEccParameter(TPM2B_ECC_PARAMETER* to,   // target
                             const TPM2B_EC_TEST* from  // source
)
{
    MemoryCopy2B(&to->b, &from->b, sizeof(to->t.buffer));
}

//*** LoadEccPoint()
static void LoadEccPoint(TPMS_ECC_POINT*      point,  // target
                         const TPM2B_EC_TEST* x,      // source
                         const TPM2B_EC_TEST* y)
{
    MemoryCopy2B(&point->x.b, (TPM2B*)x, sizeof(point->x.t.buffer));
    MemoryCopy2B(&point->y.b, (TPM2B*)y, sizeof(point->y.t.buffer));
}

//*** TestECDH()
// This test does a KVT on a point multiply.
static TPM_RC TestECDH(TPM_ALG_ID        scheme,  // IN: for consistency
```

```
                        ALGORITHM_VECTOR* toTest  // IN/OUT: modified after test is run
)
{
    static TPMS_ECC_POINT      Z;
    static TPMS_ECC_POINT      Qe;
    static TPM2B_ECC_PARAMETER ds;
    TPM_RC                     result = TPM_RC_SUCCESS;
    //
    NOT_REFERENCED(scheme);
    CLEAR_BOTH(TPM_ALG_ECDH);
    LoadEccParameter(&ds, &c_ecTestKey_ds);
    LoadEccPoint(&Qe, &c_ecTestKey_QeX, &c_ecTestKey_QeY);
    if(TPM_RC_SUCCESS != CryptEccPointMultiply(&Z, c_testCurve, &Qe, &ds, NULL, NULL))
        SELF_TEST_FAILURE;
    if(!MemoryEqual2B(&c_ecTestEcdh_X.b, &Z.x.b)
       || !MemoryEqual2B(&c_ecTestEcdh_Y.b, &Z.y.b))
        SELF_TEST_FAILURE;
    return result;
}

//*** TestEccSignAndVerify()
static TPM_RC TestEccSignAndVerify(TPM_ALG_ID scheme, ALGORITHM_VECTOR* toTest)
{
    static OBJECT          testObject;
    static TPMT_SIGNATURE  testSig;
    static TPMT_ECC_SCHEME eccScheme;

    testSig.sigAlg                    = scheme;
    testSig.signature.ecdsa.hash      = DEFAULT_TEST_HASH;

    eccScheme.scheme                  = scheme;
    eccScheme.details.anySig.hashAlg = DEFAULT_TEST_HASH;

    CLEAR_BOTH(scheme);
    CLEAR_BOTH(TPM_ALG_ECDH);

    // ECC signature verification testing uses a KVT.
    switch(scheme)
    {
        case TPM_ALG_ECDSA:
            LoadEccParameter(&testSig.signature.ecdsa.signatureR, &c_TestEcDsa_r);
            LoadEccParameter(&testSig.signature.ecdsa.signatureS, &c_TestEcDsa_s);
            break;
        case TPM_ALG_ECSCHNORR:
            LoadEccParameter(&testSig.signature.ecschnorr.signatureR,
                             &c_TestEcSchnorr_r);
            LoadEccParameter(&testSig.signature.ecschnorr.signatureS,
                             &c_TestEcSchnorr_s);
            break;
        case TPM_ALG_SM2:
            // don't have a test for SM2
            return TPM_RC_SUCCESS;
        default:
            SELF_TEST_FAILURE;
            break;
    }
    TEST_DEFAULT_TEST_HASH(toTest);

    // Have to copy the key. This is because the size used in the test vectors
    // is the size of the ECC parameter for the test key while the size of a point
    // is TPM dependent
    MemoryCopy2B(&testObject.sensitive.sensitive.ecc.b,
                 &c_ecTestKey_ds.b,
                 sizeof(testObject.sensitive.sensitive.ecc.t.buffer));
    LoadEccPoint(
        &testObject.publicArea.unique.ecc, &c_ecTestKey_QsX, &c_ecTestKey_QsY);
```

```
    testObject.publicArea.parameters.eccDetail.curveID = c_testCurve;

    if(TPM_RC_SUCCESS
       != CryptEccValidateSignature(
           &testSig, &testObject, (TPM2B_DIGEST*)&c_ecTestValue.b))
    {
        SELF_TEST_FAILURE;
    }
    CHECK_CANCELED;

    // Now sign and verify some data
    if(TPM_RC_SUCCESS
       != CryptEccSign(
           &testSig, &testObject, (TPM2B_DIGEST*)&c_ecTestValue, &eccScheme, NULL))
        SELF_TEST_FAILURE;

    CHECK_CANCELED;

    if(TPM_RC_SUCCESS
       != CryptEccValidateSignature(
           &testSig, &testObject, (TPM2B_DIGEST*)&c_ecTestValue))
        SELF_TEST_FAILURE;

    CHECK_CANCELED;

    return TPM_RC_SUCCESS;
}

//*** TestKDFa()
static TPM_RC TestKDFa(ALGORITHM_VECTOR* toTest)
{
    static TPM2B_KDF_TEST_KEY keyOut;
    UINT32                    counter = 0;
    //
    CLEAR_BOTH(TPM_ALG_KDF1_SP800_108);

    keyOut.t.size = CryptKDFa(KDF_TEST_ALG,
                              &c_kdfTestKeyIn.b,
                              &c_kdfTestLabel.b,
                              &c_kdfTestContextU.b,
                              &c_kdfTestContextV.b,
                              TEST_KDF_KEY_SIZE * 8,
                              keyOut.t.buffer,
                              &counter,
                              FALSE);
    if(keyOut.t.size != TEST_KDF_KEY_SIZE
       || !MemoryEqual(keyOut.t.buffer, c_kdfTestKeyOut.t.buffer, TEST_KDF_KEY_SIZE))
        SELF_TEST_FAILURE;

    return TPM_RC_SUCCESS;
}

//*** TestEcc()
static TPM_RC TestEcc(TPM_ALG_ID alg, ALGORITHM_VECTOR* toTest)
{
    TPM_RC result = TPM_RC_SUCCESS;
    NOT_REFERENCED(toTest);
    switch(alg)
    {
        case TPM_ALG_ECC:
        case TPM_ALG_ECDH:
            // If this is in a loop then see if another test is going to deal with
            // this.
            // If toTest is not a self-test list
            if((toTest == &g_toTest)
               // or this is the only ECC test in the list
```

```
                || !(TEST_BIT(TPM_ALG_ECDSA, *toTest)
                     || TEST_BIT(ALG_ECSCHNORR, *toTest)
                     || TEST_BIT(TPM_ALG_SM2, *toTest)))
            {
                result = TestECDH(alg, toTest);
            }
            break;
        case TPM_ALG_ECDSA:
        case TPM_ALG_ECSCHNORR:
        case TPM_ALG_SM2:
            result = TestEccSignAndVerify(alg, toTest);
            break;
        default:
            SELF_TEST_FAILURE;
            break;
    }
    return result;
}

#  endif  // ALG_ECC

//*** TestAlgorithm()
// Dispatches to the correct test function for the algorithm or gets a list of
// testable algorithms.
//
// If 'toTest' is not NULL, then the test decisions are based on the algorithm
// selections in 'toTest'. Otherwise, 'g_toTest' is used. When bits are clear in
// 'g_toTest' they will also be cleared 'toTest'.
//
// If there doesn't happen to be a test for the algorithm, its associated bit is
// quietly cleared.
//
// If 'alg' is zero (TPM_ALG_ERROR), then the toTest vector is cleared of any bits
// for which there is no test (i.e. no tests are actually run but the vector is
// cleared).
//
// Note: 'toTest' will only ever have bits set for implemented algorithms but 'alg'
// can be anything.
//
//  Return Type: TPM_RC
//      TPM_RC_CANCELED     test was canceled
LIB_EXPORT
TPM_RC
TestAlgorithm(TPM_ALG_ID alg, ALGORITHM_VECTOR* toTest)
{
    TPM_ALG_ID first  = (alg == TPM_ALG_ERROR) ? TPM_ALG_FIRST : alg;
    TPM_ALG_ID last   = (alg == TPM_ALG_ERROR) ? TPM_ALG_LAST : alg;
    BOOL       doTest = (alg != TPM_ALG_ERROR);
    TPM_RC     result = TPM_RC_SUCCESS;

    if(toTest == NULL)
        toTest = &g_toTest;

    // This is kind of strange. This function will either run a test of the selected
    // algorithm or just clear a bit if there is no test for the algorithm. So,
    // either this loop will be executed once for the selected algorithm or once for
    // each of the possible algorithms. If it is executed more than once ('alg' ==
    // ALG_ERROR), then no test will be run but bits will be cleared for
    // unimplemented algorithms. This was done this way so that there is only one
    // case statement with all of the algorithms. It was easier to have one case
    // statement than to have multiple ones to manage whenever an algorithm ID is
    // added.
    for(alg = first; (alg <= last); alg++)
    {
        // if 'alg' was TPM_ALG_ERROR, then we will be cycling through
        // values, some of which may not be implemented. If the bit in toTest
```

```
        // happens to be set, then we could either generated an assert, or just
        // silently CLEAR it. Decided to just clear.
        if(!TEST_BIT(alg, g_implementedAlgorithms))
        {
            CLEAR_BIT(alg, *toTest);
            continue;
        }
        // Process whatever is left.
        // NOTE: since this switch will only be called if the algorithm is
        // implemented, it is not necessary to modify this list except to comment
        // out the algorithms for which there is no test
        switch(alg)
        {
            // Symmetric block ciphers
#   if ALG_AES
            case TPM_ALG_AES:
#   endif  // ALG_AES
#   if ALG_SM4
                // if SM4 is implemented, its test is like other block ciphers but
there
                // aren't any test vectors for it yet
//              case TPM_ALG_SM4:
#   endif  // ALG_SM4
#   if ALG_CAMELLIA
                // no test vectors for camellia
//              case TPM_ALG_CAMELLIA:
#   endif
                // Symmetric modes
#   if !ALG_CFB
#     error CFB is required in all TPM implementations
#   endif  // !ALG_CFB
            case TPM_ALG_CFB:
                if(doTest)
                    result = TestSymmetric(alg, toTest);
                break;
#   if ALG_CTR
            case TPM_ALG_CTR:
#   endif  // ALG_CRT
#   if ALG_OFB
            case TPM_ALG_OFB:
#   endif  // ALG_OFB
#   if ALG_CBC
            case TPM_ALG_CBC:
#   endif  // ALG_CBC
#   if ALG_ECB
            case TPM_ALG_ECB:
#   endif
                if(doTest)
                    result = TestSymmetric(alg, toTest);
                else
                    // If doing the initialization of g_toTest vector, only need
                    // to test one of the modes for the symmetric algorithms. If
                    // initializing for a SelfTest(FULL_TEST), allow all the modes.
                    if(toTest == &g_toTest)
                        CLEAR_BIT(alg, *toTest);
                break;
#   if !ALG_HMAC
#     error HMAC is required in all TPM implementations
#   endif
            case TPM_ALG_HMAC:
                // Clear the bit that indicates that HMAC is required because
                // HMAC is used as the basic test for all hash algorithms.
                CLEAR_BOTH(alg);
                // Testing HMAC means test the default hash
                if(doTest)
                    TestHash(DEFAULT_TEST_HASH, toTest);
```

```
                else
                    // If not testing, then indicate that the hash needs to be
                    // tested because this uses HMAC
                    SET_BOTH(DEFAULT_TEST_HASH);
                break;
// Have to use two arguments for the macro even though only the first is used in the
// expansion.
#   define HASH_CASE_TEST(HASH, hash) case ALG_##HASH##_VALUE:
                FOR_EACH_HASH(HASH_CASE_TEST)
#   undef HASH_CASE_TEST
                if(doTest)
                    result = TestHash(alg, toTest);
                break;
                // RSA-dependent
#   if ALG_RSA
            case TPM_ALG_RSA:
                CLEAR_BOTH(alg);
                if(doTest)
                    result = TestRsa(TPM_ALG_NULL, toTest);
                else
                    SET_BOTH(TPM_ALG_NULL);
                break;
            case TPM_ALG_RSASSA:
            case TPM_ALG_RSAES:
            case TPM_ALG_RSAPSS:
            case TPM_ALG_OAEP:
            case TPM_ALG_NULL:  // used or RSADP
                if(doTest)
                    result = TestRsa(alg, toTest);
                break;
#   endif  // ALG_RSA
#   if ALG_KDF1_SP800_108
            case TPM_ALG_KDF1_SP800_108:
                if(doTest)
                    result = TestKDFa(toTest);
                break;
#   endif  // ALG_KDF1_SP800_108
#   if ALG_ECC
                // ECC dependent but no tests
                //        case TPM_ALG_ECDAA:
                //        case TPM_ALG_ECMQV:
                //        case TPM_ALG_KDF1_SP800_56a:
                //        case TPM_ALG_KDF2:
                //        case TPM_ALG_MGF1:
            case TPM_ALG_ECC:
                CLEAR_BOTH(alg);
                if(doTest)
                    result = TestEcc(TPM_ALG_ECDH, toTest);
                else
                    SET_BOTH(TPM_ALG_ECDH);
                break;
            case TPM_ALG_ECDSA:
            case TPM_ALG_ECDH:
            case TPM_ALG_ECSCHNORR:
                //            case TPM_ALG_SM2:
                if(doTest)
                    result = TestEcc(alg, toTest);
                break;
#   endif  // ALG_ECC
            default:
                CLEAR_BIT(alg, *toTest);
                break;
        }
        if(result != TPM_RC_SUCCESS)
            break;
    }
```

```
    return result;
}

#endif  // SELF_TESTS
```

7.137 /tpm/src/crypt/CryptCmac.c

```
//** Introduction
//
// This file contains the implementation of the message authentication codes based
// on a symmetric block cipher. These functions only use the single block
// encryption functions of the selected symmetric cryptographic library.

//** Includes, Defines, and Typedefs
#define _CRYPT_HASH_C_
#include "Tpm.h"
#include "CryptSym.h"

#if ALG_CMAC

//** Functions

//*** CryptCmacStart()
// This is the function to start the CMAC sequence operation. It initializes the
// dispatch functions for the data and end operations for CMAC and initializes the
// parameters that are used for the processing of data, including the key, key size
// and block cipher algorithm.
UINT16
CryptCmacStart(
    SMAC_STATE* state, TPMU_PUBLIC_PARMS* keyParms, TPM_ALG_ID macAlg, TPM2B* key)
{
    tpmCmacState_t*      cState = &state->state.cmac;
    TPMT_SYM_DEF_OBJECT* def    = &keyParms->symDetail.sym;
    //
    if(macAlg != TPM_ALG_CMAC)
        return 0;
    // set up the encryption algorithm and parameters
    cState->symAlg      = def->algorithm;
    cState->keySizeBits = def->keyBits.sym;
    cState->iv.t.size = CryptGetSymmetricBlockSize(def->algorithm, def->keyBits.sym);
    MemoryCopy2B(&cState->symKey.b, key, sizeof(cState->symKey.t.buffer));

    // Set up the dispatch methods for the CMAC
    state->smacMethods.data = CryptCmacData;
    state->smacMethods.end  = CryptCmacEnd;
    return cState->iv.t.size;
}

//*** CryptCmacData()
// This function is used to add data to the CMAC sequence computation. The function
// will XOR new data into the IV. If the buffer is full, and there is additional
// input data, the data is encrypted into the IV buffer, the new data is then
// XOR into the IV. When the data runs out, the function returns without encrypting
// even if the buffer is full. The last data block of a sequence will not be
// encrypted until the call to CryptCmacEnd(). This is to allow the proper subkey
// to be computed and applied before the last block is encrypted.
void CryptCmacData(SMAC_STATES* state, UINT32 size, const BYTE* buffer)
{
    tpmCmacState_t*         cmacState     = &state->cmac;
    TPM_ALG_ID              algorithm     = cmacState->symAlg;
    BYTE*                   key           = cmacState->symKey.t.buffer;
    UINT16                  keySizeInBits = cmacState->keySizeBits;
    tpmCryptKeySchedule_t   keySchedule;
    TpmCryptSetSymKeyCall_t encrypt;
    //
```

```
    // Set up the encryption values based on the algorithm
    switch(algorithm)
    {
        FOR_EACH_SYM(ENCRYPT_CASE)
        default:
            FAIL(FATAL_ERROR_INTERNAL);
    }
    while(size > 0)
    {
        if(cmacState->bcount == cmacState->iv.t.size)
        {
            ENCRYPT(&keySchedule, cmacState->iv.t.buffer, cmacState->iv.t.buffer);
            cmacState->bcount = 0;
        }
        for(; (size > 0) && (cmacState->bcount < cmacState->iv.t.size);
            size--, cmacState->bcount++)
        {
            cmacState->iv.t.buffer[cmacState->bcount] ^= *buffer++;
        }
    }
}

//*** CryptCmacEnd()
// This is the completion function for the CMAC. It does padding, if needed, and
// selects the subkey to be applied before the last block is encrypted.
UINT16
CryptCmacEnd(SMAC_STATES* state, UINT32 outSize, BYTE* outBuffer)
{
    tpmCmacState_t* cState = &state->cmac;
    // Need to set algorithm, key, and keySizeInBits in the local context so that
    // the SELECT and ENCRYPT macros will work here
    TPM_ALG_ID              algorithm     = cState->symAlg;
    BYTE*                   key           = cState->symKey.t.buffer;
    UINT16                  keySizeInBits = cState->keySizeBits;
    tpmCryptKeySchedule_t   keySchedule;
    TpmCryptSetSymKeyCall_t encrypt;
    TPM2B_IV                subkey = {{0, {0}}};
    BOOL                    xorVal;
    UINT16                  i;

    subkey.t.size = cState->iv.t.size;
    // Encrypt a block of zero
    // Set up the encryption values based on the algorithm
    switch(algorithm)
    {
        FOR_EACH_SYM(ENCRYPT_CASE)
        default:
            return 0;
    }
    ENCRYPT(&keySchedule, subkey.t.buffer, subkey.t.buffer);

    // shift left by 1 and XOR with 0x0...87 if the MSb was 0
    xorVal = ((subkey.t.buffer[0] & 0x80) == 0) ? 0 : 0x87;
    ShiftLeft(&subkey.b);
    subkey.t.buffer[subkey.t.size - 1] ^= xorVal;
    // this is a sanity check to make sure that the algorithm is working properly.
    // remove this check when debug is done
    pAssert(cState->bcount <= cState->iv.t.size);
    // If the buffer is full then no need to compute subkey 2.
    if(cState->bcount < cState->iv.t.size)
    {
        //Pad the data
        cState->iv.t.buffer[cState->bcount++] ^= 0x80;
        // The rest of the data is a pad of zero which would simply be XORed
        // with the iv value so nothing to do...
        // Now compute K2
```

```
        xorVal = ((subkey.t.buffer[0] & 0x80) == 0) ? 0 : 0x87;
        ShiftLeft(&subkey.b);
        subkey.t.buffer[subkey.t.size - 1] ^= xorVal;
    }
    // XOR the subkey into the IV
    for(i = 0; i < subkey.t.size; i++)
        cState->iv.t.buffer[i] ^= subkey.t.buffer[i];
    ENCRYPT(&keySchedule, cState->iv.t.buffer, cState->iv.t.buffer);
    i = (UINT16)MIN(cState->iv.t.size, outSize);
    MemoryCopy(outBuffer, cState->iv.t.buffer, i);

    return i;
}
#endif
```

7.138 /tpm/src/crypt/CryptEccCrypt.c

```
//** Includes and Defines
#include "Tpm.h"
#include "TpmMath_Util_fp.h"
#include "TpmEcc_Util_fp.h"

#if CC_ECC_Encrypt || CC_ECC_Encrypt

//** Functions

//*** CryptEccSelectScheme()
// This function is used by TPM2_ECC_Decrypt and TPM2_ECC_Encrypt.  It sets scheme
// either the input scheme or the key scheme. If they key scheme is not TPM_ALG_NULL
// then the input scheme must be TPM_ALG_NULL or the same as the key scheme. If
// not, then the function returns FALSE.
//  Return Type: BOOL
//      TRUE        'scheme' is set
//      FALSE       'scheme' is not valid (it may have been changed).
BOOL CryptEccSelectScheme(OBJECT*          key,    //IN: key containing default scheme
                          TPMT_KDF_SCHEME* scheme  // IN: a decrypt scheme
)
{
    TPMT_KDF_SCHEME* keyScheme = &key->publicArea.parameters.eccDetail.kdf;

    // Get sign object pointer
    if(scheme->scheme == TPM_ALG_NULL)
        *scheme = *keyScheme;
    if(keyScheme->scheme == TPM_ALG_NULL)
        keyScheme = scheme;
    return (
        scheme->scheme != TPM_ALG_NULL
        && (keyScheme->scheme == scheme->scheme
            && keyScheme->details.anyKdf.hashAlg == scheme->details.anyKdf.hashAlg));
}

//*** CryptEccEncrypt()
//This function performs ECC-based data obfuscation. The only scheme that is currently
// supported is MGF1 based. See Part 1, Annex D for details.
//  Return Type: TPM_RC
//      TPM_RC_CURVE            unsupported curve
//      TPM_RC_HASH             hash not allowed
//      TPM_RC_SCHEME           'scheme' is not supported
//      TPM_RC_NO_RESULT        internal error in big number processing
LIB_EXPORT TPM_RC CryptEccEncrypt(
    OBJECT*           key,        // IN: public key of recipient
    TPMT_KDF_SCHEME*  scheme,     // IN: scheme to use.
    TPM2B_MAX_BUFFER* plainText,  // IN: the text to obfuscate
    TPMS_ECC_POINT*   c1,         // OUT: public ephemeral key
    TPM2B_MAX_BUFFER* c2,         // OUT: obfuscated text
```

```
    TPM2B_DIGEST*     c3          // OUT: digest of ephemeral key
                                  //      and plainText
)
{
    CRYPT_CURVE_INITIALIZED(E, key->publicArea.parameters.eccDetail.curveID);
    CRYPT_POINT_INITIALIZED(PB, &key->publicArea.unique.ecc);
    CRYPT_POINT_VAR(Px);
    TPMS_ECC_POINT p2;
    CRYPT_ECC_NUM(D);
    TPM2B_TYPE(2ECC, MAX_ECC_KEY_BYTES * 2);
    TPM2B_2ECC z;
    int        i;
    HASH_STATE hashState;
    TPM_RC     retVal = TPM_RC_SUCCESS;
    //
#   if defined DEBUG_ECC_ENCRYPT && DEBUG_ECC_ENCRYPT == YES
    RND_DEBUG dbg;
    // This value is one less than the value from the reference so that it
    // will become the correct value after having one added
    TPM2B_ECC_PARAMETER k = {24, {0x38, 0x4F, 0x30, 0x35, 0x30, 0x73, 0xAE, 0xEC,
                                  0xE7, 0xA1, 0x65, 0x43, 0x30, 0xA9, 0x62, 0x04,
                                  0xD3, 0x79, 0x82, 0xA3, 0xE1, 0x5B, 0x2C, 0xB4}};
    RND_DEBUG_Instantiate(&dbg, &k.b);
#    define RANDOM (RAND_STATE*)&dbg

#   else
#    define RANDOM NULL
#   endif
    if(E == NULL)
        ERROR_EXIT(TPM_RC_CURVE);
    if(TPM_ALG_KDF2 != scheme->scheme)
        ERROR_EXIT(TPM_RC_SCHEME);
    // generate an ephemeral key from a random k
    if(!TpmEcc_GenerateKeyPair(D, Px, E, RANDOM)
       // C1 is the public part of the ephemeral key
       || !TpmEcc_PointTo2B(c1, Px, E)
       // Compute P2
       || (TpmEcc_PointMult(Px, PB, D, NULL, NULL, E) != TPM_RC_SUCCESS)
       || !TpmEcc_PointTo2B(&p2, Px, E))
        ERROR_EXIT(TPM_RC_NO_RESULT);

    //Compute the C3 value hash(x2 || M || y2)
    if(0 == CryptHashStart(&hashState, scheme->details.mgf1.hashAlg))
        ERROR_EXIT(TPM_RC_HASH);
    CryptDigestUpdate2B(&hashState, &p2.x.b);
    CryptDigestUpdate2B(&hashState, &plainText->b);
    CryptDigestUpdate2B(&hashState, &p2.y.b);
    c3->t.size = CryptHashEnd(&hashState, sizeof(c3->t.buffer), c3->t.buffer);

    MemoryCopy2B(&z.b, &p2.x.b, sizeof(z.t.buffer));
    MemoryConcat2B(&z.b, &p2.y.b, sizeof(z.t.buffer));
    // Generate the mask value from MGF1 and put it in the return buffer
    c2->t.size = CryptMGF_KDF(plainText->t.size,
                              c2->t.buffer,
                              scheme->details.mgf1.hashAlg,
                              z.t.size,
                              z.t.buffer,
                              1);
    // XOR the plainText into the generated mask to create the obfuscated data
    for(i = 0; i < plainText->t.size; i++)
        c2->t.buffer[i] ^= plainText->t.buffer[i];
Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}
```

```
//*** CryptEccDecrypt()
// This function performs ECC decryption and integrity check of the input data.
//  Return Type: TPM_RC
//      TPM_RC_CURVE            unsupported curve
//      TPM_RC_HASH             hash not allowed
//      TPM_RC_SCHEME           'scheme' is not supported
//      TPM_RC_NO_RESULT        internal error in big number processing
//      TPM_RC_VALUE            C3 did not match hash of recovered data
LIB_EXPORT TPM_RC CryptEccDecrypt(
    OBJECT*           key,        // IN: key used for data recovery
    TPMT_KDF_SCHEME*  scheme,     // IN: scheme to use.
    TPM2B_MAX_BUFFER* plainText,  // OUT: the recovered text
    TPMS_ECC_POINT*   c1,         // IN: public ephemeral key
    TPM2B_MAX_BUFFER* c2,         // IN: obfuscated text
    TPM2B_DIGEST*     c3          // IN: digest of ephemeral key
                                  //      and plainText
)
{
    CRYPT_CURVE_INITIALIZED(E, key->publicArea.parameters.eccDetail.curveID);
    CRYPT_ECC_INITIALIZED(D, &key->sensitive.sensitive.ecc.b);
    CRYPT_POINT_INITIALIZED(C1, c1);
    TPMS_ECC_POINT p2;
    TPM2B_TYPE(2ECC, MAX_ECC_KEY_BYTES * 2);
    TPM2B_DIGEST check;
    TPM2B_2ECC   z;
    int          i;
    HASH_STATE   hashState;
    TPM_RC       retVal = TPM_RC_SUCCESS;
    //
    if(E == NULL)
        ERROR_EXIT(TPM_RC_CURVE);
    if(TPM_ALG_KDF2 != scheme->scheme)
        ERROR_EXIT(TPM_RC_SCHEME);
    // Generate the Z value
    TpmEcc_PointMult(C1, C1, D, NULL, NULL, E);
    TpmEcc_PointTo2B(&p2, C1, E);

    // Start the hash to check the algorithm
    if(0 == CryptHashStart(&hashState, scheme->details.mgf1.hashAlg))
        ERROR_EXIT(TPM_RC_HASH);
    CryptDigestUpdate2B(&hashState, &p2.x.b);

    MemoryCopy2B(&z.b, &p2.x.b, sizeof(z.t.buffer));
    MemoryConcat2B(&z.b, &p2.y.b, sizeof(z.t.buffer));

    // Generate the mask
    plainText->t.size = CryptMGF_KDF(c2->t.size,
                                     plainText->t.buffer,
                                     scheme->details.mgf1.hashAlg,
                                     z.t.size,
                                     z.t.buffer,
                                     1);
    // XOR the obfuscated data into the generated mask to create the plainText data
    for(i = 0; i < plainText->t.size; i++)
        plainText->t.buffer[i] ^= c2->t.buffer[i];

    // Complete the hash and verify the data
    CryptDigestUpdate2B(&hashState, &plainText->b);
    CryptDigestUpdate2B(&hashState, &p2.y.b);
    check.t.size = CryptHashEnd(&hashState, sizeof(check.t.buffer), check.t.buffer);
    if(!MemoryEqual2B(&check.b, &c3->b))
        ERROR_EXIT(TPM_RC_VALUE);
Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}
```

```
#endif  // CC_ECC_Encrypt || CC_ECC_Encrypt
```

7.139 /tpm/src/crypt/CryptEccData.c

```
/*(Auto-generated)
 *  Created by TpmStructures; Version 4.4 Mar 26, 2019
 *  Date: Aug 30, 2019  Time: 02:11:52PM
 */

#include "Tpm.h"
#include "OIDs.h"

#if ALG_ECC

// This file contains the TPM Specific ECC curve metadata and pointers to the ecc-lib
specific
// constant structure.
// The CURVE_NAME macro is used to remove the name string from normal builds, but
leaves the
// string available in the initialization lists for potenial use during debugging by
changing this
// macro (and the structure declaration)
#  define CURVE_NAME(N)

#  define comma
const TPM_ECC_CURVE_METADATA eccCurves[] = {
#  if ECC_NIST_P192
    comma{TPM_ECC_NIST_P192,
          192,
          {TPM_ALG_KDF1_SP800_56A, {{TPM_ALG_SHA256}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_NIST_P192 CURVE_NAME("NIST_P192")}
#    undef comma
#    define comma ,
#  endif  // ECC_NIST_P192
#  if ECC_NIST_P224
    comma{TPM_ECC_NIST_P224,
          224,
          {TPM_ALG_KDF1_SP800_56A, {{TPM_ALG_SHA256}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_NIST_P224 CURVE_NAME("NIST_P224")}
#    undef comma
#    define comma ,
#  endif  // ECC_NIST_P224
#  if ECC_NIST_P256
    comma{TPM_ECC_NIST_P256,
          256,
          {TPM_ALG_KDF1_SP800_56A, {{TPM_ALG_SHA256}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_NIST_P256 CURVE_NAME("NIST_P256")}
#    undef comma
#    define comma ,
#  endif  // ECC_NIST_P256
#  if ECC_NIST_P384
    comma{TPM_ECC_NIST_P384,
          384,
          {TPM_ALG_KDF1_SP800_56A, {{TPM_ALG_SHA384}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_NIST_P384 CURVE_NAME("NIST_P384")}
#    undef comma
#    define comma ,
#  endif  // ECC_NIST_P384
#  if ECC_NIST_P521
    comma{TPM_ECC_NIST_P521,
```

```
          521,
          {TPM_ALG_KDF1_SP800_56A, {{TPM_ALG_SHA512}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_NIST_P521 CURVE_NAME("NIST_P521")}
#    undef comma
#    define comma ,
#  endif  // ECC_NIST_P521
#  if ECC_BN_P256
    comma{TPM_ECC_BN_P256,
          256,
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_BN_P256 CURVE_NAME("BN_P256")}
#    undef comma
#    define comma ,
#  endif  // ECC_BN_P256
#  if ECC_BN_P638
    comma{TPM_ECC_BN_P638,
          638,
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_BN_P638 CURVE_NAME("BN_P638")}
#    undef comma
#    define comma ,
#  endif  // ECC_BN_P638
#  if ECC_SM2_P256
    comma{TPM_ECC_SM2_P256,
          256,
          {TPM_ALG_KDF1_SP800_56A, {{TPM_ALG_SM3_256}}},
          {TPM_ALG_NULL, {{TPM_ALG_NULL}}},
          OID_ECC_SM2_P256 CURVE_NAME("SM2_P256")}
#    undef comma
#    define comma ,
#  endif  // ECC_SM2_P256
};

#endif  // TPM_ALG_ECC
```

7.140 /tpm/src/crypt/CryptEccKeyExchange.c

```
//** Introduction
// This file contains the functions that are used for the two-phase, ECC,
// key-exchange protocols

#include "Tpm.h"
#include "TpmMath_Util_fp.h"
#include "TpmEcc_Util_fp.h"

#if CC_ZGen_2Phase == YES

//** Functions

#  if ALG_ECMQV

//*** avf1()
// This function does the associated value computation required by MQV key
// exchange.
// Process:
// 1. Convert 'xQ' to an integer 'xqi' using the convention specified in Appendix C.3.
// 2. Calculate
//        xqm = xqi mod 2^ceil(f/2) (where f = ceil(log2(n)).
// 3. Calculate the associate value function
//        avf(Q) = xqm + 2ceil(f / 2)
// Always returns TRUE(1).
static BOOL avf1(Crypt_Int* bnX,  // IN/OUT: the reduced value
```

```
                Crypt_Int* bnN   // IN: the order of the curve
)
{
    // compute f = 2^(ceil(ceil(log2(n)) / 2))
    int f = (ExtMath_SizeInBits(bnN) + 1) / 2;
    // x' = 2^f + (x mod 2^f)
    ExtMath_MaskBits(bnX, f);  // This is mod 2*2^f but it doesn't matter because
        // the next operation will SET the extra bit anyway
    if(!ExtMath_SetBit(bnX, f))
    {
        FAIL(FATAL_ERROR_CRYPTO);
    }
    return TRUE;
}

//*** C_2_2_MQV()
// This function performs the key exchange defined in SP800-56A
// 6.1.1.4 Full MQV, C(2, 2, ECC MQV).
//
// CAUTION: Implementation of this function may require use of essential claims in
// patents not owned by TCG members.
//
// Points 'QsB' and 'QeB' are required to be on the curve of 'inQsA'. The function
// will fail, possibly catastrophically, if this is not the case.
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        the value for dsA does not give a valid point on the
//                              curve
static TPM_RC C_2_2_MQV(TPMS_ECC_POINT* outZ,   // OUT: the computed point
                        TPM_ECC_CURVE curveId,  // IN: the curve for the computations
                        TPM2B_ECC_PARAMETER* dsA,  // IN: static private TPM key
                        TPM2B_ECC_PARAMETER* deA,  // IN: ephemeral private TPM key
                        TPMS_ECC_POINT*      QsB,  // IN: static public party B key
                        TPMS_ECC_POINT*      QeB   // IN: ephemeral public party B key
)
{
    CRYPT_CURVE_INITIALIZED(E, curveId);
    CRYPT_POINT_VAR(pQeA);
    CRYPT_POINT_INITIALIZED(pQeB, QeB);
    CRYPT_POINT_INITIALIZED(pQsB, QsB);
    CRYPT_ECC_NUM(bnTa);
    CRYPT_ECC_INITIALIZED(bnDeA, deA);
    CRYPT_ECC_INITIALIZED(bnDsA, dsA);
    CRYPT_ECC_NUM(bnN);
    CRYPT_ECC_NUM(bnXeB);
    TPM_RC retVal;
    //
    // Parameter checks
    if(E == NULL)
        ERROR_EXIT(TPM_RC_VALUE);
    pAssert(
        outZ != NULL && pQeB != NULL && pQsB != NULL && deA != NULL && dsA != NULL);
    // Process:
    //  1. implicitsigA = (de,A + avf(Qe,A)ds,A ) mod n.
    //  2. P = h(implicitsigA)(Qe,B + avf(Qe,B)Qs,B).
    //  3. If P = O, output an error indicator.
    //  4. Z=xP, where xP is the x-coordinate of P.

    // Compute the public ephemeral key pQeA = [de,A]G
    if((retVal =
            TpmEcc_PointMult(pQeA, ExtEcc_CurveGetG(curveId), bnDeA, NULL, NULL, E))
       != TPM_RC_SUCCESS)
        goto Exit;

    //  1. implicitsigA = (de,A + avf(Qe,A)ds,A ) mod n.
    //  tA := (ds,A + de,A  avf(Xe,A)) mod n     (3)
    //  Compute 'tA' = ('deA' +  'dsA'  avf('XeA')) mod n
```

```
    // Ta = avf(XeA);
    ExtMath_Copy(bnTa, ExtEcc_PointX(pQeA));
    avf1(bnTa, bnN);
    // do Ta = ds,A * Ta mod n = dsA * avf(XeA) mod n
    ExtMath_ModMult(bnTa, bnDsA, bnTa, bnN);
    // now Ta = deA + Ta mod n =  deA + dsA * avf(XeA) mod n
    ExtMath_Add(bnTa, bnTa, bnDeA);
    ExtMath_Mod(bnTa, bnN);

    //  2. P = h(implicitsigA)(Qe,B + avf(Qe,B)Qs,B).
    // Put this in because almost every case of h is == 1 so skip the call when
    // not necessary.
    if(!ExtMath_IsEqualWord(ExtEcc_CurveGetCofactor(curveId), 1))
        // Cofactor is not 1 so compute Ta := Ta * h mod n
        ExtMath_ModMult(bnTa,
                        bnTa,
                        ExtEcc_CurveGetCofactor(curveId),
                        ExtEcc_CurveGetOrder(curveId));

    // Now that 'tA' is (h * 'tA' mod n)
    // 'outZ' = (tA)(Qe,B + avf(Qe,B)Qs,B).

    // first, compute XeB = avf(XeB)
    avf1(bnXeB, bnN);

    // QsB := [XeB]QsB
    TpmEcc_PointMult(pQsB, pQsB, bnXeB, NULL, NULL, E);
    ExtEcc_PointAdd(pQeB, pQeB, pQsB, E);

    // QeB := [tA]QeB = [tA](QsB + [Xe,B]QeB) and check for at infinity
    // If the result is not the point at infinity, return QeB
    TpmEcc_PointMult(pQeB, pQeB, bnTa, NULL, NULL, E);
    if(ExtEcc_IsInfinityPoint(pQeB))
        ERROR_EXIT(TPM_RC_NO_RESULT);
    // Convert Crypt_Int* E to TPM2B E
    TpmEcc_PointTo2B(outZ, pQeB, E);

Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}

#  endif  // ALG_ECMQV

//*** C_2_2_ECDH()
// This function performs the two phase key exchange defined in SP800-56A,
// 6.1.1.2 Full Unified Model, C(2, 2, ECC CDH).
//
static TPM_RC C_2_2_ECDH(TPMS_ECC_POINT* outZs,  // OUT: Zs
                         TPMS_ECC_POINT* outZe,  // OUT: Ze
                         TPM_ECC_CURVE curveId,  // IN: the curve for the computations
                         TPM2B_ECC_PARAMETER* dsA,  // IN: static private TPM key
                         TPM2B_ECC_PARAMETER* deA,  // IN: ephemeral private TPM key
                         TPMS_ECC_POINT*      QsB,  // IN: static public party B key
                         TPMS_ECC_POINT*      QeB  // IN: ephemeral public party B key
)
{
    CRYPT_CURVE_INITIALIZED(E, curveId);
    CRYPT_ECC_INITIALIZED(bnAs, dsA);
    CRYPT_ECC_INITIALIZED(bnAe, deA);
    CRYPT_POINT_INITIALIZED(ecBs, QsB);
    CRYPT_POINT_INITIALIZED(ecBe, QeB);
    CRYPT_POINT_VAR(ecZ);
    TPM_RC retVal;
    //
    // Parameter checks
```

```
    if(E == NULL)
        ERROR_EXIT(TPM_RC_CURVE);
    pAssert(
        outZs != NULL && dsA != NULL && deA != NULL && QsB != NULL && QeB != NULL);

    // Do the point multiply for the Zs value ([dsA]QsB)
    retVal = TpmEcc_PointMult(ecZ, ecBs, bnAs, NULL, NULL, E);
    if(retVal == TPM_RC_SUCCESS)
    {
        // Convert the Zs value.
        TpmEcc_PointTo2B(outZs, ecZ, E);
        // Do the point multiply for the Ze value ([deA]QeB)
        retVal = TpmEcc_PointMult(ecZ, ecBe, bnAe, NULL, NULL, E);
        if(retVal == TPM_RC_SUCCESS)
            TpmEcc_PointTo2B(outZe, ecZ, E);
    }
Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}

//*** CryptEcc2PhaseKeyExchange()
// This function is the dispatch routine for the EC key exchange functions that use
// two ephemeral and two static keys.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME             scheme is not defined
LIB_EXPORT TPM_RC CryptEcc2PhaseKeyExchange(
    TPMS_ECC_POINT*      outZ1,    // OUT: a computed point
    TPMS_ECC_POINT*      outZ2,    // OUT: and optional second point
    TPM_ECC_CURVE        curveId,  // IN: the curve for the computations
    TPM_ALG_ID           scheme,   // IN: the key exchange scheme
    TPM2B_ECC_PARAMETER* dsA,      // IN: static private TPM key
    TPM2B_ECC_PARAMETER* deA,      // IN: ephemeral private TPM key
    TPMS_ECC_POINT*      QsB,      // IN: static public party B key
    TPMS_ECC_POINT*      QeB       // IN: ephemeral public party B key
)
{
    pAssert(
        outZ1 != NULL && dsA != NULL && deA != NULL && QsB != NULL && QeB != NULL);

    // Initialize the output points so that they are empty until one of the
    // functions decides otherwise
    outZ1->x.b.size = 0;
    outZ1->y.b.size = 0;
    if(outZ2 != NULL)
    {
        outZ2->x.b.size = 0;
        outZ2->y.b.size = 0;
    }
    switch(scheme)
    {
        case TPM_ALG_ECDH:
            return C_2_2_ECDH(outZ1, outZ2, curveId, dsA, deA, QsB, QeB);
            break;
#   if ALG_ECMQV
        case TPM_ALG_ECMQV:
            return C_2_2_MQV(outZ1, curveId, dsA, deA, QsB, QeB);
            break;
#   endif
#   if ALG_SM2
        case TPM_ALG_SM2:
            return SM2KeyExchange(outZ1, curveId, dsA, deA, QsB, QeB);
            break;
#   endif
        default:
            return TPM_RC_SCHEME;
```

```
    }
}

#  if ALG_SM2

//*** ComputeWForSM2()
// Compute the value for w used by SM2
static UINT32 ComputeWForSM2(TPM_ECC_CURVE curveId)
{
    //  w := ceil(ceil(log2(n)) / 2) - 1
    return (ExtMath_MostSigBitNum(ExtEcc_CurveGetOrder(curveId)) / 2 - 1);
}

//*** avfSm2()
// This function does the associated value computation required by SM2 key
// exchange. This is different from the avf() in the international standards
// because it returns a value that is half the size of the value returned by the
// standard avf(). For example, if 'n' is 15, 'Ws' ('w' in the standard) is 2 but
// the 'W' here is 1. This means that an input value of 14 (1110b) would return a
// value of 110b with the standard but 10b with the scheme in SM2.
static Crypt_Int* avfSm2(Crypt_Int* bn,  // IN/OUT: the reduced value
                         UINT32     w    // IN: the value of w
)
{
    // a)   set w := ceil(ceil(log2(n)) / 2) - 1
    // b)   set x' := 2^w + ( x & (2^w - 1))
    // This is just like the avf for MQV where x' = 2^w + (x mod 2^w)

    ExtMath_MaskBits(bn, w);  // as with avf1, this is too big by a factor of 2 but
                              // it doesn't matter because we SET the extra bit
                              // anyway
    if(!ExtMath_SetBit(bn, w))
    {
        FAIL(FATAL_ERROR_CRYPTO);
    }
    return bn;
}

//*** SM2KeyExchange()
// This function performs the key exchange defined in SM2.
// The first step is to compute
//  'tA' = ('dsA' + 'deA'  avf(Xe,A)) mod 'n'
// Then, compute the 'Z' value from
// 'outZ' = ('h'  'tA' mod 'n') ('QsA' + [avf('QeB.x')]('QeB')).
// The function will compute the ephemeral public key from the ephemeral
// private key.
// All points are required to be on the curve of 'inQsA'. The function will fail
// catastrophically if this is not the case
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        the value for dsA does not give a valid point on the
//                              curve
LIB_EXPORT TPM_RC SM2KeyExchange(
    TPMS_ECC_POINT*      outZ,     // OUT: the computed point
    TPM_ECC_CURVE        curveId,  // IN: the curve for the computations
    TPM2B_ECC_PARAMETER* dsAIn,    // IN: static private TPM key
    TPM2B_ECC_PARAMETER* deAIn,    // IN: ephemeral private TPM key
    TPMS_ECC_POINT*      QsBIn,    // IN: static public party B key
    TPMS_ECC_POINT*      QeBIn     // IN: ephemeral public party B key
)
{
    CRYPT_CURVE_INITIALIZED(E, curveId);
    CRYPT_ECC_INITIALIZED(dsA, dsAIn);
    CRYPT_ECC_INITIALIZED(deA, deAIn);
    CRYPT_POINT_INITIALIZED(QsB, QsBIn);
    CRYPT_POINT_INITIALIZED(QeB, QeBIn);
    CRYPT_INT_WORD_INITIALIZED(One, 1);
```

```
    CRYPT_POINT_VAR(QeA);
    CRYPT_ECC_NUM(XeB);
    CRYPT_POINT_VAR(Z);
    CRYPT_ECC_NUM(Ta);
    CRYPT_ECC_NUM(QeA_X);
    UINT32 w;
    TPM_RC retVal = TPM_RC_NO_RESULT;
    //
    // Parameter checks
    if(E == NULL)
        ERROR_EXIT(TPM_RC_CURVE);
    pAssert(outZ != NULL && dsA != NULL && deA != NULL && QsB != NULL && QeB != NULL);

    // Compute the value for w
    w = ComputeWForSM2(curveId);

    // Compute the public ephemeral key pQeA = [de,A]G
    if(!ExtEcc_PointMultiply(QeA, ExtEcc_CurveGetG(curveId), deA, E))
        goto Exit;

    //  tA := (ds,A + de,A  avf(Xe,A)) mod n    (3)
    //  Compute 'tA' = ('dsA' +  'deA'  avf('XeA')) mod n
    // Ta = avf(XeA);
    // do Ta = de,A * Ta = deA * avf(XeA)
    ExtMath_Copy(QeA_X, ExtEcc_PointX(QeA));  // create mutable copy
    ExtMath_Multiply(Ta, deA, avfSm2(QeA_X, w));
    // now Ta = dsA + Ta =  dsA + deA * avf(XeA)
    ExtMath_Add(Ta, dsA, Ta);
    ExtMath_Mod(Ta, ExtEcc_CurveGetOrder(curveId));

    //  outZ = [h  tA mod n] (Qs,B + [avf(Xe,B)](Qe,B)) (4)
    // Put this in because almost every case of h is == 1 so skip the call when
    // not necessary.
    if(!ExtMath_IsEqualWord(ExtEcc_CurveGetCofactor(curveId), 1))
        // Cofactor is not 1 so compute Ta := Ta * h mod n
        ExtMath_ModMult(
            Ta, Ta, ExtEcc_CurveGetCofactor(curveId), ExtEcc_CurveGetOrder(curveId));
    // Now that 'tA' is (h * 'tA' mod n)
    // 'outZ' = ['tA'](QsB + [avf(QeB.x)](QeB)).
    ExtMath_Copy(XeB, ExtEcc_PointX(QeB));
    if(!ExtEcc_PointMultiplyAndAdd(Z, QsB, One, QeB, avfSm2(XeB, w), E))
        goto Exit;
    // QeB := [tA]QeB = [tA](QsB + [Xe,B]QeB) and check for at infinity
    if(!ExtEcc_PointMultiply(Z, Z, Ta, E))
        goto Exit;
    // Convert Crypt_Int* E to TPM2B E
    TpmEcc_PointTo2B(outZ, Z, E);
    retVal = TPM_RC_SUCCESS;
Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}
#  endif

#endif  // CC_ZGen_2Phase
```

7.141 /tpm/src/crypt/CryptEccMain.c

```
//** Includes and Defines
#include "Tpm.h"
#include "TpmMath_Util_fp.h"
#include "TpmEcc_Util_fp.h"
#include "TpmEcc_Signature_ECDSA_fp.h"  // required for pairwise test in key
generation
#if ALG_ECC
```

```
//** Functions

#  if SIMULATION
void EccSimulationEnd(void)
{
#    if SIMULATION
// put things to be printed at the end of the simulation here
#    endif
}
#  endif  // SIMULATION

//*** CryptEccInit()
// This function is called at _TPM_Init
BOOL CryptEccInit(void)
{
    return TRUE;
}

//*** CryptEccStartup()
// This function is called at TPM2_Startup().
BOOL CryptEccStartup(void)
{
    return TRUE;
}

//*** ClearPoint2B(generic)
// Initialize the size values of a TPMS_ECC_POINT structure.
void ClearPoint2B(TPMS_ECC_POINT* p  // IN: the point
)
{
    if(p != NULL)
    {
        p->x.t.size = 0;
        p->y.t.size = 0;
    }
}

//*** CryptEccGetParametersByCurveId()
// This function returns a pointer to the curve data that is associated with
// the indicated curveId.
// If there is no curve with the indicated ID, the function returns NULL. This
// function is in this module so that it can be called by GetCurve data.
//  Return Type: const TPM_ECC_CURVE_METADATA
//      NULL            curve with the indicated TPM_ECC_CURVE is not implemented
//      != NULL         pointer to the curve data
LIB_EXPORT const TPM_ECC_CURVE_METADATA* CryptEccGetParametersByCurveId(
    TPM_ECC_CURVE curveId  // IN: the curveID
)
{
    int i;
    for(i = 0; i < ECC_CURVE_COUNT; i++)
    {
        if(eccCurves[i].curveId == curveId)
            return &eccCurves[i];
    }
    return NULL;
}

//*** CryptEccGetKeySizeForCurve()
// This function returns the key size in bits of the indicated curve.
LIB_EXPORT UINT16 CryptEccGetKeySizeForCurve(TPM_ECC_CURVE curveId  // IN: the curve
)
{
    const TPM_ECC_CURVE_METADATA* curve = CryptEccGetParametersByCurveId(curveId);
    UINT16                        keySizeInBits;
    //
```

```
    keySizeInBits = (curve != NULL) ? curve->keySizeBits : 0;
    return keySizeInBits;
}

//***CryptEccGetOID()
const BYTE* CryptEccGetOID(TPM_ECC_CURVE curveId)
{
    const TPM_ECC_CURVE_METADATA* curve = CryptEccGetParametersByCurveId(curveId);
    return (curve != NULL) ? curve->OID : NULL;
}

//*** CryptEccGetCurveByIndex()
// This function returns the number of the 'i'-th implemented curve. The normal
// use would be to call this function with 'i' starting at 0. When the 'i' is greater
// than or equal to the number of implemented curves, TPM_ECC_NONE is returned.
LIB_EXPORT TPM_ECC_CURVE CryptEccGetCurveByIndex(UINT16 i)
{
    if(i >= ECC_CURVE_COUNT)
        return TPM_ECC_NONE;
    return eccCurves[i].curveId;
}

//*** CryptCapGetECCCurve()
// This function returns the list of implemented ECC curves.
//  Return Type: TPMI_YES_NO
//      YES             if no more ECC curve is available
//      NO              if there are more ECC curves not reported
TPMI_YES_NO
CryptCapGetECCCurve(TPM_ECC_CURVE   curveID,   // IN: the starting ECC curve
                    UINT32          maxCount,  // IN: count of returned curves
                    TPML_ECC_CURVE* curveList  // OUT: ECC curve list
)
{
    TPMI_YES_NO   more = NO;
    UINT16        i;
    UINT32        count = ECC_CURVE_COUNT;
    TPM_ECC_CURVE curve;

    // Initialize output property list
    curveList->count = 0;

    // The maximum count of curves we may return is MAX_ECC_CURVES
    if(maxCount > MAX_ECC_CURVES)
        maxCount = MAX_ECC_CURVES;

    // Scan the eccCurveValues array
    for(i = 0; i < count; i++)
    {
        curve = CryptEccGetCurveByIndex(i);
        // If curveID is less than the starting curveID, skip it
        if(curve < curveID)
            continue;
        if(curveList->count < maxCount)
        {
            // If we have not filled up the return list, add more curves to
            // it
            curveList->eccCurves[curveList->count] = curve;
            curveList->count++;
        }
        else
        {
            // If the return list is full but we still have curves
            // available, report this and stop iterating
            more = YES;
            break;
        }
```

```
    }
    return more;
}

//*** CryptCapGetOneECCCurve()
// This function returns whether the ECC curve is implemented.
BOOL CryptCapGetOneECCCurve(TPM_ECC_CURVE curveID  // IN: the  ECC curve
)
{
    UINT16 i;

    // Scan the eccCurveValues array
    for(i = 0; i < ECC_CURVE_COUNT; i++)
    {
        if(CryptEccGetCurveByIndex(i) == curveID)
        {
            return TRUE;
        }
    }
    return FALSE;
}

//*** CryptGetCurveSignScheme()
// This function will return a pointer to the scheme of the curve.
const TPMT_ECC_SCHEME* CryptGetCurveSignScheme(
    TPM_ECC_CURVE curveId  // IN: The curve selector
)
{
    const TPM_ECC_CURVE_METADATA* curve = CryptEccGetParametersByCurveId(curveId);

    if(curve != NULL)
        return &(curve->sign);
    else
        return NULL;
}

//*** CryptGenerateR()
// This function computes the commit random value for a split signing scheme.
//
// If 'c' is NULL, it indicates that 'r' is being generated
// for TPM2_Commit.
// If 'c' is not NULL, the TPM will validate that the 'gr.commitArray'
// bit associated with the input value of 'c' is SET. If not, the TPM
// returns FALSE and no 'r' value is generated.
//  Return Type: BOOL
//      TRUE(1)         r value computed
//      FALSE(0)        no r value computed
BOOL CryptGenerateR(TPM2B_ECC_PARAMETER* r,        // OUT: the generated random value
                    UINT16*              c,        // IN/OUT: count value.
                    TPMI_ECC_CURVE       curveID,  // IN: the curve for the value
                    TPM2B_NAME*          name      // IN: optional name of a key to
                                                   //     associate with 'r'
)
{
    // This holds the marshaled g_commitCounter.
    TPM2B_TYPE(8B, 8);
    TPM2B_8B            cntr = {{8, {0}}};
    UINT32              iterations;
    TPM2B_ECC_PARAMETER n;
    UINT64              currentCount = gr.commitCounter;
    UINT16              t1;
    //
    if(!TpmMath_IntTo2B(ExtEcc_CurveGetOrder(curveID), (TPM2B*)&n, 0))
        return FALSE;

    // If this is the commit phase, use the current value of the commit counter
```

```
    if(c != NULL)
    {
        // if the array bit is not set, can't use the value.
        if(!TEST_BIT((*c & COMMIT_INDEX_MASK), gr.commitArray))
            return FALSE;

        // If it is the sign phase, figure out what the counter value was
        // when the commitment was made.
        //
        // When gr.commitArray has less than 64K bits, the extra
        // bits of 'c' are used as a check to make sure that the
        // signing operation is not using an out of range count value
        t1 = (UINT16)currentCount;

        // If the lower bits of c are greater or equal to the lower bits of t1
        // then the upper bits of t1 must be one more than the upper bits
        // of c
        if((*c & COMMIT_INDEX_MASK) >= (t1 & COMMIT_INDEX_MASK))
            // Since the counter is behind, reduce the current count
            currentCount = currentCount - (COMMIT_INDEX_MASK + 1);

        t1 = (UINT16)currentCount;
        if((t1 & ~COMMIT_INDEX_MASK) != (*c & ~COMMIT_INDEX_MASK))
            return FALSE;
        // set the counter to the value that was
        // present when the commitment was made
        currentCount = (currentCount & 0xffffffffffff0000) | *c;
    }
    // Marshal the count value to a TPM2B buffer for the KDF
    cntr.t.size = sizeof(currentCount);
    UINT64_TO_BYTE_ARRAY(currentCount, cntr.t.buffer);

    // Now can do the KDF to create the random value for the signing operation
    // During the creation process, we may generate an r that does not meet the
    // requirements of the random value.
    // want to generate a new r.
    r->t.size = n.t.size;

    for(iterations = 1; iterations < 1000000;)
    {
        int i;
        CryptKDFa(CONTEXT_INTEGRITY_HASH_ALG,
                  &gr.commitNonce.b,
                  COMMIT_STRING,
                  &name->b,
                  &cntr.b,
                  n.t.size * 8,
                  r->t.buffer,
                  &iterations,
                  FALSE);

        // "random" value must be less than the prime
        if(UnsignedCompareB(r->b.size, r->b.buffer, n.t.size, n.t.buffer) >= 0)
            continue;

        // in this implementation it is required that at least bit
        // in the upper half of the number be set
        for(i = n.t.size / 2; i >= 0; i--)
            if(r->b.buffer[i] != 0)
                return TRUE;
    }
    return FALSE;
}

//*** CryptCommit()
// This function is called when the count value is committed. The 'gr.commitArray'
```

```
// value associated with the current count value is SET and g_commitCounter is
// incremented. The low-order 16 bits of old value of the counter is returned.
UINT16
CryptCommit(void)
{
    UINT16 oldCount = (UINT16)gr.commitCounter;
    gr.commitCounter++;
    SET_BIT(oldCount & COMMIT_INDEX_MASK, gr.commitArray);
    return oldCount;
}

//*** CryptEndCommit()
// This function is called when the signing operation using the committed value
// is completed. It clears the gr.commitArray bit associated with the count
// value so that it can't be used again.
void CryptEndCommit(UINT16 c  // IN: the counter value of the commitment
)
{
    ClearBit((c & COMMIT_INDEX_MASK), gr.commitArray, sizeof(gr.commitArray));
}

//*** CryptEccGetParameters()
// This function returns the ECC parameter details of the given curve.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        unsupported ECC curve ID
BOOL CryptEccGetParameters(
    TPM_ECC_CURVE               curveId,    // IN: ECC curve ID
    TPMS_ALGORITHM_DETAIL_ECC* parameters  // OUT: ECC parameters
)
{
    const TPM_ECC_CURVE_METADATA* curve = CryptEccGetParametersByCurveId(curveId);
    BOOL                          found = curve != NULL;

    if(found)
    {
        parameters->curveID = curve->curveId;
        parameters->keySize = curve->keySizeBits;
        parameters->kdf     = curve->kdf;
        parameters->sign    = curve->sign;
        //        BnTo2B(data->prime, &parameters->p.b, 0);
        found = found
                && TpmMath_IntTo2B(ExtEcc_CurveGetPrime(curveId),
                                   &parameters->p.b,
                                   parameters->p.t.size);
        found = found
                && TpmMath_IntTo2B(ExtEcc_CurveGet_a(curveId), &parameters->a.b, 0);
        found = found
                && TpmMath_IntTo2B(ExtEcc_CurveGet_b(curveId), &parameters->b.b, 0);
        found = found
                && TpmMath_IntTo2B(ExtEcc_CurveGetGx(curveId),
                                   &parameters->gX.b,
                                   parameters->p.t.size);
        found = found
                && TpmMath_IntTo2B(ExtEcc_CurveGetGy(curveId),
                                   &parameters->gY.b,
                                   parameters->p.t.size);
        //        BnTo2B(data->base.x, &parameters->gX.b, 0);
        //        BnTo2B(data->base.y, &parameters->gY.b, 0);
        found =
            found
            && TpmMath_IntTo2B(ExtEcc_CurveGetOrder(curveId), &parameters->n.b, 0);
        found =
            found
            && TpmMath_IntTo2B(ExtEcc_CurveGetCofactor(curveId), &parameters->h.b, 0);
        // if we got into this IF but failed to get a parameter from the external
```

```
        // library, our crypto systems are broken; enter failure mode.
        if(!found)
        {
            FAIL(FATAL_ERROR_MATHLIBRARY);
        }
    }
    return found;
}

//*** TpmEcc_IsValidPrivateEcc()
// Checks that 0 < 'x' < 'q'
BOOL TpmEcc_IsValidPrivateEcc(const Crypt_Int*      x,  // IN: private key to check
                              const Crypt_EccCurve* E   // IN: the curve to check
)
{
    BOOL retVal;
    retVal =
        (!ExtMath_IsZero(x)
         && (ExtMath_UnsignedCmp(x, ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E)))
             < 0));
    return retVal;
}

LIB_EXPORT BOOL CryptEccIsValidPrivateKey(TPM2B_ECC_PARAMETER* d,
                                          TPM_ECC_CURVE        curveId)
{
    CRYPT_INT_INITIALIZED(bnD, MAX_ECC_PARAMETER_BYTES * 8, d);
    return !ExtMath_IsZero(bnD)
           && (ExtMath_UnsignedCmp(bnD, ExtEcc_CurveGetOrder(curveId)) < 0);
}

//*** TpmEcc_PointMult()
// This function does a point multiply of the form 'R' = ['d']'S' + ['u']'Q' where the
// parameters are Crypt_Int* values. If 'S' is NULL and d is not NULL, then it
computes
// 'R' = ['d']'G' + ['u']'Q'  or just 'R' = ['d']'G' if 'u' and 'Q' are NULL.
// If 'skipChecks' is TRUE, then the function will not verify that the inputs are
// correct for the domain. This would be the case when the values were created by the
// CryptoEngine code.
// It will return TPM_RC_NO_RESULT if the resulting point is the point at infinity.
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        result of multiplication is a point at infinity
//      TPM_RC_ECC_POINT        'S' or 'Q' is not on the curve
//      TPM_RC_VALUE            'd' or 'u' is not < n
TPM_RC
TpmEcc_PointMult(Crypt_Point*          R,  // OUT: computed point
                 const Crypt_Point*    S,  // IN: optional point to multiply by 'd'
                 const Crypt_Int*      d,  // IN: scalar for [d]S or [d]G
                 const Crypt_Point*    Q,  // IN: optional second point
                 const Crypt_Int*      u,  // IN: optional second scalar
                 const Crypt_EccCurve* E   // IN: curve parameters
)
{
    BOOL OK;
    //
    TPM_DO_SELF_TEST(TPM_ALG_ECDH);

    // Need one scalar
    OK = (d != NULL || u != NULL);

    // If S is present, then d has to be present. If S is not
    // present, then d may or may not be present
    OK = OK && (((S == NULL) == (d == NULL)) || (d != NULL));

    // either both u and Q have to be provided or neither can be provided (don't
    // know what to do if only one is provided.
```

```
    OK = OK && ((u == NULL) == (Q == NULL));

    OK = OK && (E != NULL);
    if(!OK)
        return TPM_RC_VALUE;

    OK = (S == NULL) || ExtEcc_IsPointOnCurve(S, E);
    OK = OK && ((Q == NULL) || ExtEcc_IsPointOnCurve(Q, E));
    if(!OK)
        return TPM_RC_ECC_POINT;

    if((d != NULL) && (S == NULL))
        S = ExtEcc_CurveGetG(ExtEcc_CurveGetCurveId(E));
    // If only one scalar, don't need Shamir's trick
    if((d == NULL) || (u == NULL))
    {
        if(d == NULL)
            OK = ExtEcc_PointMultiply(R, Q, u, E);
        else
            OK = ExtEcc_PointMultiply(R, S, d, E);
    }
    else
    {
        OK = ExtEcc_PointMultiplyAndAdd(R, S, d, Q, u, E);
    }
    return (OK ? TPM_RC_SUCCESS : TPM_RC_NO_RESULT);
}

//***TpmEcc_GenPrivateScalar()
// This function gets random values that are the size of the key plus 64 bits. The
// value is reduced (mod ('q' - 1)) and incremented by 1 ('q' is the order of the
// curve. This produces a value ('d') such that 1 <= 'd' < 'q'. This is the method
// of FIPS 186-4 Section B.4.1 ""Key Pair Generation Using Extra Random Bits"".
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure generating private key
BOOL TpmEcc_GenPrivateScalar(
    Crypt_Int*            dOut,  // OUT: the qualified random value
    const Crypt_EccCurve* E,     // IN: curve for which the private key
                                 //     needs to be appropriate
    RAND_STATE* rand             // IN: state for DRBG
)
{
    TPM_ECC_CURVE    curveId = ExtEcc_CurveGetCurveId(E);
    const Crypt_Int* order   = ExtEcc_CurveGetOrder(curveId);
    BOOL             OK;
    UINT32           orderBits  = ExtMath_SizeInBits(order);
    UINT32           orderBytes = BITS_TO_BYTES(orderBits);
    CRYPT_INT_VAR(bnExtraBits, MAX_ECC_KEY_BITS + 64);
    CRYPT_INT_VAR(nMinus1, MAX_ECC_KEY_BITS);
    //
    OK = TpmMath_GetRandomInteger(bnExtraBits, (orderBytes * 8) + 64, rand);
    OK = OK && ExtMath_SubtractWord(nMinus1, order, 1);
    OK = OK && ExtMath_Mod(bnExtraBits, nMinus1);
    OK = OK && ExtMath_AddWord(dOut, bnExtraBits, 1);
    return OK && !g_inFailureMode;
}

//*** TpmEcc_GenerateKeyPair()
// This function gets a private scalar from the source of random bits and does
// the point multiply to get the public key.
BOOL TpmEcc_GenerateKeyPair(Crypt_Int*            bnD,  // OUT: private scalar
                            Crypt_Point*          ecQ,  // OUT: public point
                            const Crypt_EccCurve* E,    // IN: curve for the point
                            RAND_STATE*           rand  // IN: DRBG state to use
)
```

```
{
    BOOL OK = FALSE;
    // Get a private scalar
    OK = TpmEcc_GenPrivateScalar(bnD, E, rand);

    // Do a point multiply
    OK = OK && ExtEcc_PointMultiply(ecQ, NULL, bnD, E);
    return OK;
}

//***CryptEccNewKeyPair(***)
// This function creates an ephemeral ECC. It is ephemeral in that
// is expected that the private part of the key will be discarded
LIB_EXPORT TPM_RC CryptEccNewKeyPair(
    TPMS_ECC_POINT*      Qout,    // OUT: the public point
    TPM2B_ECC_PARAMETER* dOut,    // OUT: the private scalar
    TPM_ECC_CURVE        curveId  // IN: the curve for the key
)
{
    CRYPT_CURVE_INITIALIZED(E, curveId);
    CRYPT_POINT_VAR(ecQ);
    CRYPT_ECC_NUM(bnD);
    BOOL OK;

    if(E == NULL)
        return TPM_RC_CURVE;

    TPM_DO_SELF_TEST(TPM_ALG_ECDH);
    OK = TpmEcc_GenerateKeyPair(bnD, ecQ, E, NULL);
    if(OK)
    {
        TpmEcc_PointTo2B(Qout, ecQ, E);
        TpmMath_IntTo2B(bnD, &dOut->b, Qout->x.t.size);
    }
    else
    {
        Qout->x.t.size = Qout->y.t.size = dOut->t.size = 0;
    }
    CRYPT_CURVE_FREE(E);
    return OK ? TPM_RC_SUCCESS : TPM_RC_NO_RESULT;
}

//*** CryptEccPointMultiply()
// This function computes 'R' := ['dIn']'G' + ['uIn']'QIn'. Where 'dIn' and
// 'uIn' are scalars, 'G' and 'QIn' are points on the specified curve and 'G' is the
// default generator of the curve.
//
// The 'xOut' and 'yOut' parameters are optional and may be set to NULL if not
// used.
//
// It is not necessary to provide 'uIn' if 'QIn' is specified but one of 'uIn' and
// 'dIn' must be provided. If 'dIn' and 'QIn' are specified but 'uIn' is not
// provided, then 'R' = ['dIn']'QIn'.
//
// If the multiply produces the point at infinity, the TPM_RC_NO_RESULT is returned.
//
// The sizes of 'xOut' and yOut' will be set to be the size of the degree of
// the curve
//
// It is a fatal error if 'dIn' and 'uIn' are both unspecified (NULL) or if 'Qin'
// or 'Rout' is unspecified.
//
//  Return Type: TPM_RC
//      TPM_RC_ECC_POINT         the point 'Pin' or 'Qin' is not on the curve
//      TPM_RC_NO_RESULT         the product point is at infinity
//      TPM_RC_CURVE             bad curve
```

```
//      TPM_RC_VALUE            'dIn' or 'uIn' out of range
//
LIB_EXPORT TPM_RC CryptEccPointMultiply(
    TPMS_ECC_POINT*      Rout,     // OUT: the product point R
    TPM_ECC_CURVE        curveId,  // IN: the curve to use
    TPMS_ECC_POINT*      Pin,      // IN: first point (can be null)
    TPM2B_ECC_PARAMETER* dIn,      // IN: scalar value for [dIn]Qin
                                   //     the Pin
    TPMS_ECC_POINT*      Qin,      // IN: point Q
    TPM2B_ECC_PARAMETER* uIn       // IN: scalar value for the multiplier
                                   //     of Q
)
{
    CRYPT_CURVE_INITIALIZED(E, curveId);
    CRYPT_POINT_INITIALIZED(ecP, Pin);
    CRYPT_ECC_INITIALIZED(bnD, dIn);  // If dIn is null, then bnD is null
    CRYPT_ECC_INITIALIZED(bnU, uIn);
    CRYPT_POINT_INITIALIZED(ecQ, Qin);
    CRYPT_POINT_VAR(ecR);
    TPM_RC retVal;
    //
    retVal = TpmEcc_PointMult(ecR, ecP, bnD, ecQ, bnU, E);

    if(retVal == TPM_RC_SUCCESS)
        TpmEcc_PointTo2B(Rout, ecR, E);
    else
        ClearPoint2B(Rout);
    CRYPT_CURVE_FREE(E);
    return retVal;
}

//*** CryptEccIsPointOnCurve()
// This function is used to test if a point is on a defined curve. It does this
// by checking that 'y'^2 mod 'p' = 'x'^3 + 'a'*'x' + 'b' mod 'p'.
//
// It is a fatal error if 'Q' is not specified (is NULL).
//  Return Type: BOOL
//      TRUE(1)         point is on curve
//      FALSE(0)        point is not on curve or curve is not supported
LIB_EXPORT BOOL CryptEccIsPointOnCurve(
    TPM_ECC_CURVE   curveId,  // IN: the curve selector
    TPMS_ECC_POINT* Qin       // IN: the point.
)
{
    CRYPT_CURVE_INITIALIZED(E, curveId);
    CRYPT_POINT_INITIALIZED(ecQ, Qin);
    BOOL OK;
    //
    pAssert(Qin != NULL);
    OK = (E != NULL && (ExtEcc_IsPointOnCurve(ecQ, E)));
    return OK;
}

//*** CryptEccGenerateKey()
// This function generates an ECC key pair based on the input parameters.
// This routine uses KDFa to produce candidate numbers. The method is according
// to FIPS 186-3, section B.1.2 "Key Pair Generation by Testing Candidates."
// According to the method in FIPS 186-3, the resulting private value 'd' should be
// 1 <= 'd' < 'n' where 'n' is the order of the base point.
//
// It is a fatal error if 'Qout', 'dOut', is not provided (is NULL).
//
// If the curve is not supported
// If 'seed' is not provided, then a random number will be used for the key
//  Return Type: TPM_RC
//      TPM_RC_CURVE            curve is not supported
```

```
//      TPM_RC_NO_RESULT        could not verify key with signature (FIPS only)
LIB_EXPORT TPM_RC CryptEccGenerateKey(
    TPMT_PUBLIC* publicArea,    // IN/OUT: The public area template for
                                //      the new key. The public key
                                //      area will be replaced computed
                                //      ECC public key
    TPMT_SENSITIVE* sensitive,  // OUT: the sensitive area will be
                                //      updated to contain the private
                                //      ECC key and the symmetric
                                //      encryption key
    RAND_STATE* rand            // IN: if not NULL, the deterministic
                                //     RNG state
)
{
    CRYPT_CURVE_INITIALIZED(E, publicArea->parameters.eccDetail.curveID);
    CRYPT_ECC_NUM(bnD);
    CRYPT_POINT_VAR(ecQ);
    BOOL   OK;
    TPM_RC retVal;
    //
    TPM_DO_SELF_TEST(TPM_ALG_ECDSA);  // ECDSA is used to verify each key

    // Validate parameters
    if(E == NULL)
        ERROR_EXIT(TPM_RC_CURVE);

    publicArea->unique.ecc.x.t.size = 0;
    publicArea->unique.ecc.y.t.size = 0;
    sensitive->sensitive.ecc.t.size = 0;

    OK                              = TpmEcc_GenerateKeyPair(bnD, ecQ, E, rand);
    if(OK)
    {
        TpmEcc_PointTo2B(&publicArea->unique.ecc, ecQ, E);
        TpmMath_IntTo2B(
            bnD, &sensitive->sensitive.ecc.b, publicArea->unique.ecc.x.t.size);
    }
#  if FIPS_COMPLIANT
    // See if PWCT is required
    if(OK && IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
    {
        CRYPT_ECC_NUM(bnT);
        CRYPT_ECC_NUM(bnS);
        TPM2B_DIGEST digest;
        //
        TPM_DO_SELF_TEST(TPM_ALG_ECDSA);
        digest.t.size = MIN(sensitive->sensitive.ecc.t.size, sizeof(digest.t.buffer));
        // Get a random value to sign using the built in DRBG state
        DRBG_Generate(NULL, digest.t.buffer, digest.t.size);
        if(g_inFailureMode)
            return TPM_RC_FAILURE;
        TpmEcc_SignEcdsa(bnT, bnS, E, bnD, &digest, NULL);
        // and make sure that we can validate the signature
        OK = TpmEcc_ValidateSignatureEcdsa(bnT, bnS, E, ecQ, &digest)
             == TPM_RC_SUCCESS;
    }
#  endif
    retVal = (OK) ? TPM_RC_SUCCESS : TPM_RC_NO_RESULT;
Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}

#endif  // ALG_ECC
```

7.142 /tpm/src/crypt/CryptEccSignature.c

```
//** Includes and Defines
#include "Tpm.h"
#include "TpmEcc_Signature_ECDSA_fp.h"
#include "TpmEcc_Signature_ECDAA_fp.h"
#include "TpmEcc_Signature_Schnorr_fp.h"
#include "TpmEcc_Signature_SM2_fp.h"
#include "TpmEcc_Util_fp.h"
#include "TpmMath_Util_fp.h"
#include "CryptEccSignature_fp.h"

#if ALG_ECC

//** Utility Functions

//** Signing Functions

//*** CryptEccSign()
// This function is the dispatch function for the various ECC-based
// signing schemes.
// There is a bit of ugliness to the parameter passing. In order to test this,
// we sometime would like to use a deterministic RNG so that we can get the same
// signatures during testing. The easiest way to do this for most schemes is to
// pass in a deterministic RNG and let it return canned values during testing.
// There is a competing need for a canned parameter to use in ECDAA. To accommodate
// both needs with minimal fuss, a special type of RAND_STATE is defined to carry
// the address of the commit value. The setup and handling of this is not very
// different for the caller than what was in previous versions of the code.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME           'scheme' is not supported
LIB_EXPORT TPM_RC CryptEccSign(TPMT_SIGNATURE* signature,  // OUT: signature
                               OBJECT* signKey,  // IN: ECC key to sign the hash
                               const TPM2B_DIGEST* digest,  // IN: digest to sign
                               TPMT_ECC_SCHEME*    scheme,  // IN: signing scheme
                               RAND_STATE*         rand)
{
    CRYPT_CURVE_INITIALIZED(E, signKey->publicArea.parameters.eccDetail.curveID);
    CRYPT_ECC_INITIALIZED(bnD, &signKey->sensitive.sensitive.ecc.b);
    CRYPT_ECC_NUM(bnR);
    CRYPT_ECC_NUM(bnS);
    TPM_RC retVal = TPM_RC_SCHEME;
    //
    NOT_REFERENCED(scheme);
    if(E == NULL)
        ERROR_EXIT(TPM_RC_VALUE);
    signature->signature.ecdaa.signatureR.t.size =
        sizeof(signature->signature.ecdaa.signatureR.t.buffer);
    signature->signature.ecdaa.signatureS.t.size =
        sizeof(signature->signature.ecdaa.signatureS.t.buffer);
    TPM_DO_SELF_TEST(signature->sigAlg);
    switch(signature->sigAlg)
    {
        case TPM_ALG_ECDSA:
            retVal = TpmEcc_SignEcdsa(bnR, bnS, E, bnD, digest, rand);
            break;
#   if ALG_ECDAA
        case TPM_ALG_ECDAA:
            retVal = TpmEcc_SignEcdaa(&signature->signature.ecdaa.signatureR,
                                      bnS,
                                      E,
                                      bnD,
                                      digest,
                                      scheme,
                                      signKey,
                                      rand);
```

```
            bnR    = NULL;
            break;
#   endif
#   if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            retVal = TpmEcc_SignEcSchnorr(
                bnR, bnS, E, bnD, digest, signature->signature.ecschnorr.hash, rand);
            break;
#   endif
#   if ALG_SM2
        case TPM_ALG_SM2:
            retVal = TpmEcc_SignEcSm2(bnR, bnS, E, bnD, digest, rand);
            break;
#   endif
        default:
            break;
    }
    // If signature generation worked, convert the results.
    if(retVal == TPM_RC_SUCCESS)
    {
        NUMBYTES orderBytes = (NUMBYTES)BITS_TO_BYTES(
            ExtMath_SizeInBits(ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E))));
        if(bnR != NULL)
            TpmMath_IntTo2B(
                bnR, &signature->signature.ecdaa.signatureR.b, orderBytes);
        if(bnS != NULL)
            TpmMath_IntTo2B(
                bnS, &signature->signature.ecdaa.signatureS.b, orderBytes);
    }
Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}

//********************** Signature Validation   ********************

//*** CryptEccValidateSignature()
// This function validates an EcDsa or EcSchnorr signature.
// The point 'Qin' needs to have been validated to be on the curve of 'curveId'.
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE            not a valid signature
LIB_EXPORT TPM_RC CryptEccValidateSignature(
    TPMT_SIGNATURE*     signature,  // IN: signature to be verified
    OBJECT*             signKey,    // IN: ECC key signed the hash
    const TPM2B_DIGEST* digest      // IN: digest that was signed
)
{
    CRYPT_CURVE_INITIALIZED(E, signKey->publicArea.parameters.eccDetail.curveID);
    CRYPT_ECC_NUM(bnR);
    CRYPT_ECC_NUM(bnS);
    CRYPT_POINT_INITIALIZED(ecQ, &signKey->publicArea.unique.ecc);
    const Crypt_Int* order;
    TPM_RC           retVal;

    if(E == NULL)
        ERROR_EXIT(TPM_RC_VALUE);

    order = ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E));

    //    // Make sure that the scheme is valid
    switch(signature->sigAlg)
    {
        case TPM_ALG_ECDSA:
#   if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
#   endif
```

```
#   if ALG_SM2
        case TPM_ALG_SM2:
#   endif
            break;
        default:
            ERROR_EXIT(TPM_RC_SCHEME);
            break;
    }
    // Can convert r and s after determining that the scheme is an ECC scheme. If
    // this conversion doesn't work, it means that the unmarshaling code for
    // an ECC signature is broken.
    TpmMath_IntFrom2B(bnR, &signature->signature.ecdsa.signatureR.b);
    TpmMath_IntFrom2B(bnS, &signature->signature.ecdsa.signatureS.b);

    // r and s have to be greater than 0 but less than the curve order
    if(ExtMath_IsZero(bnR) || ExtMath_IsZero(bnS))
        ERROR_EXIT(TPM_RC_SIGNATURE);
    if((ExtMath_UnsignedCmp(bnS, order) >= 0)
       || (ExtMath_UnsignedCmp(bnR, order) >= 0))
        ERROR_EXIT(TPM_RC_SIGNATURE);

    switch(signature->sigAlg)
    {
        case TPM_ALG_ECDSA:
            retVal = TpmEcc_ValidateSignatureEcdsa(bnR, bnS, E, ecQ, digest);
            break;

#   if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            retVal = TpmEcc_ValidateSignatureEcSchnorr(
                bnR, bnS, signature->signature.any.hashAlg, E, ecQ, digest);
            break;
#   endif
#   if ALG_SM2
        case TPM_ALG_SM2:
            retVal = TpmEcc_ValidateSignatureEcSm2(bnR, bnS, E, ecQ, digest);
            break;
#   endif
        default:
            FAIL(FATAL_ERROR_INTERNAL);
    }
Exit:
    CRYPT_CURVE_FREE(E);
    return retVal;
}

//***CryptEccCommitCompute()
// This function performs the point multiply operations required by TPM2_Commit.
//
// If 'B' or 'M' is provided, they must be on the curve defined by 'curveId'. This
// routine does not check that they are on the curve and results are unpredictable
// if they are not.
//
// It is a fatal error if 'r' is NULL. If 'B' is not NULL, then it is a
// fatal error if 'd' is NULL or if 'K' and 'L' are both NULL.
// If 'M' is not NULL, then it is a fatal error if 'E' is NULL.
//
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        if 'K', 'L' or 'E' was computed to be the point
//                              at infinity
//      TPM_RC_CANCELED         a cancel indication was asserted during this
//                              function
LIB_EXPORT TPM_RC CryptEccCommitCompute(
    TPMS_ECC_POINT*      K,         // OUT: [d]B or [r]Q
    TPMS_ECC_POINT*      L,         // OUT: [r]B
    TPMS_ECC_POINT*      E,         // OUT: [r]M
```

```
    TPM_ECC_CURVE        curveId,  // IN: the curve for the computations
    TPMS_ECC_POINT*      M,        // IN: M (optional)
    TPMS_ECC_POINT*      B,        // IN: B (optional)
    TPM2B_ECC_PARAMETER* d,        // IN: d (optional)
    TPM2B_ECC_PARAMETER* r         // IN: the computed r value (required)
)
{
    // Normally initialize E as the curve, but
    // E means something else in this function
    CRYPT_CURVE_INITIALIZED(curve, curveId);
    CRYPT_ECC_INITIALIZED(bnR, r);
    TPM_RC retVal = TPM_RC_SUCCESS;
    //
    // Validate that the required parameters are provided.
    // Note: E has to be provided if computing E := [r]Q or E := [r]M. Will do
    // E := [r]Q if both M and B are NULL.
    pAssert(r != NULL && E != NULL);

    // Initialize the output points in case they are not computed
    ClearPoint2B(K);
    ClearPoint2B(L);
    ClearPoint2B(E);

    // Sizes of the r parameter may not be zero
    pAssert(r->t.size > 0);

    // If B is provided, compute K=[d]B and L=[r]B
    if(B != NULL)
    {
        CRYPT_ECC_INITIALIZED(bnD, d);
        CRYPT_POINT_INITIALIZED(pB, B);
        CRYPT_POINT_VAR(pK);
        CRYPT_POINT_VAR(pL);
        //
        pAssert(d != NULL && K != NULL && L != NULL);

        if(!ExtEcc_IsPointOnCurve(pB, curve))
            ERROR_EXIT(TPM_RC_VALUE);
        // do the math for K = [d]B
        if((retVal = TpmEcc_PointMult(pK, pB, bnD, NULL, NULL, curve))
           != TPM_RC_SUCCESS)
            goto Exit;
        // Convert BN K to TPM2B K
        TpmEcc_PointTo2B(K, pK, curve);
        //  compute L= [r]B after checking for cancel
        if(_plat__IsCanceled())
            ERROR_EXIT(TPM_RC_CANCELED);
        // compute L = [r]B
        if(!TpmEcc_IsValidPrivateEcc(bnR, curve))
            ERROR_EXIT(TPM_RC_VALUE);
        if((retVal = TpmEcc_PointMult(pL, pB, bnR, NULL, NULL, curve))
           != TPM_RC_SUCCESS)
            goto Exit;
        // Convert BN L to TPM2B L
        TpmEcc_PointTo2B(L, pL, curve);
    }
    if((M != NULL) || (B == NULL))
    {
        CRYPT_POINT_INITIALIZED(pM, M);
        CRYPT_POINT_VAR(pE);
        //
        // Make sure that a place was provided for the result
        pAssert(E != NULL);

        // if this is the third point multiply, check for cancel first
        if((B != NULL) && _plat__IsCanceled())
```

```
            ERROR_EXIT(TPM_RC_CANCELED);

        // If M provided, then pM will not be NULL and will compute E = [r]M.
        // However, if M was not provided, then pM will be NULL and E = [r]G
        // will be computed
        if((retVal = TpmEcc_PointMult(pE, pM, bnR, NULL, NULL, curve))
           != TPM_RC_SUCCESS)
            goto Exit;
        // Convert E to 2B format
        TpmEcc_PointTo2B(E, pE, curve);
    }
Exit:
    CRYPT_CURVE_FREE(curve);
    return retVal;
}

#endif  // ALG_ECC
```

7.143 /tpm/src/crypt/CryptHash.c

```
//** Description
//
// This file contains implementation of cryptographic functions for hashing.
//
//** Includes, Defines, and Types

#define _CRYPT_HASH_C_
#include "Tpm.h"
#include "CryptHash_fp.h"
#include "CryptHash.h"
#include "OIDs.h"

// Instance each of the hash descriptors based on the implemented algorithms
FOR_EACH_HASH(HASH_DEF_TEMPLATE)
// Instance a 'null' def.
HASH_DEF NULL_Def = {{0}};

// Create a table of pointers to the defined hash definitions
#define HASH_DEF_ENTRY(HASH, Hash) &Hash##_Def,
PHASH_DEF HashDefArray[] = {
    // for each implemented HASH, expands to: &HASH_Def,
    FOR_EACH_HASH(HASH_DEF_ENTRY) & NULL_Def};
#undef HASH_DEF_ENTRY

//** Obligatory Initialization Functions

//*** CryptHashInit()
// This function is called by _TPM_Init do perform the initialization operations for
// the library.
BOOL CryptHashInit(void)
{
    LibHashInit();
    return TRUE;
}

//*** CryptHashStartup()
// This function is called by TPM2_Startup(). It checks that the size of the
// HashDefArray is consistent with the HASH_COUNT.
BOOL CryptHashStartup(void)
{
    int i = sizeof(HashDefArray) / sizeof(PHASH_DEF) - 1;
    return (i == HASH_COUNT);
}

//** Hash Information Access Functions
```

```
//*** Introduction
// These functions provide access to the hash algorithm description information.

//*** CryptGetHashDef()
// This function accesses the hash descriptor associated with a hash a
// algorithm. The function returns a pointer to a 'null' descriptor if hashAlg is
// TPM_ALG_NULL or not a defined algorithm.
PHASH_DEF
CryptGetHashDef(TPM_ALG_ID hashAlg)
{
#define GET_DEF(HASH, Hash)  \
    case ALG_##HASH##_VALUE: \
        return &Hash##_Def;
    switch(hashAlg)
    {
        FOR_EACH_HASH(GET_DEF)
        default:
            return &NULL_Def;
    }
#undef GET_DEF
}

//*** CryptHashIsValidAlg()
// This function tests to see if an algorithm ID is a valid hash algorithm. If
// flag is true, then TPM_ALG_NULL is a valid hash.
//  Return Type: BOOL
//      TRUE(1)         hashAlg is a valid, implemented hash on this TPM
//      FALSE(0)        hashAlg is not valid for this TPM
BOOL CryptHashIsValidAlg(TPM_ALG_ID hashAlg,  // IN: the algorithm to check
                         BOOL       flag  // IN: TRUE if TPM_ALG_NULL is to be treated
                                          //     as a valid hash
)
{
    if(hashAlg == TPM_ALG_NULL)
        return flag;
    return CryptGetHashDef(hashAlg) != &NULL_Def;
}

//*** CryptHashGetAlgByIndex()
// This function is used to iterate through the hashes. TPM_ALG_NULL
// is returned for all indexes that are not valid hashes.
// If the TPM implements 3 hashes, then an 'index' value of 0 will
// return the first implemented hash and an 'index' of 2 will return the
// last. All other index values will return TPM_ALG_NULL.
//
//  Return Type: TPM_ALG_ID
// TPM_ALG_xxx          a hash algorithm
// TPM_ALG_NULL         this can be used as a stop value
LIB_EXPORT TPM_ALG_ID CryptHashGetAlgByIndex(UINT32 index  // IN: the index
)
{
    TPM_ALG_ID hashAlg;
    if(index >= HASH_COUNT)
        hashAlg = TPM_ALG_NULL;
    else
        hashAlg = HashDefArray[index]->hashAlg;
    return hashAlg;
}

//*** CryptHashGetDigestSize()
// Returns the size of the digest produced by the hash. If 'hashAlg' is not a hash
// algorithm, the TPM will FAIL.
//  Return Type: UINT16
//   0       TPM_ALG_NULL
//   > 0     the digest size
//
```

```
LIB_EXPORT UINT16 CryptHashGetDigestSize(
    TPM_ALG_ID hashAlg  // IN: hash algorithm to look up
)
{
    return CryptGetHashDef(hashAlg)->digestSize;
}

//*** CryptHashGetBlockSize()
// Returns the size of the block used by the hash. If 'hashAlg' is not a hash
// algorithm, the TPM will FAIL.
//  Return Type: UINT16
//   0       TPM_ALG_NULL
//   > 0     the digest size
//
LIB_EXPORT UINT16 CryptHashGetBlockSize(
    TPM_ALG_ID hashAlg  // IN: hash algorithm to look up
)
{
    return CryptGetHashDef(hashAlg)->blockSize;
}

//*** CryptHashGetOid()
// This function returns a pointer to DER=encoded OID for a hash algorithm. All OIDs
// are full OID values including the Tag (0x06) and length byte.
LIB_EXPORT const BYTE* CryptHashGetOid(TPM_ALG_ID hashAlg)
{
    return CryptGetHashDef(hashAlg)->OID;
}

//***  CryptHashGetContextAlg()
// This function returns the hash algorithm associated with a hash context.
TPM_ALG_ID
CryptHashGetContextAlg(PHASH_STATE state  // IN: the context to check
)
{
    return state->hashAlg;
}

//** State Import and Export

//*** CryptHashCopyState
// This function is used to clone a HASH_STATE.
LIB_EXPORT void CryptHashCopyState(HASH_STATE* out,  // OUT: destination of the state
                                   const HASH_STATE* in  // IN: source of the state
)
{
    pAssert(out->type == in->type);
    out->hashAlg = in->hashAlg;
    out->def     = in->def;
    if(in->hashAlg != TPM_ALG_NULL)
    {
        HASH_STATE_COPY(out, in);
    }
    if(in->type == HASH_STATE_HMAC)
    {
        const HMAC_STATE* hIn  = (HMAC_STATE*)in;
        HMAC_STATE*       hOut = (HMAC_STATE*)out;
        hOut->hmacKey          = hIn->hmacKey;
    }
    return;
}

//*** CryptHashExportState()
// This function is used to export a hash or HMAC hash state. This function
// would be called when preparing to context save a sequence object.
void CryptHashExportState(
```

```
    PCHASH_STATE internalFmt,       // IN: the hash state formatted for use by
                                    //     library
    PEXPORT_HASH_STATE externalFmt  // OUT: the exported hash state
)
{
    BYTE* outBuf = (BYTE*)externalFmt;
    //
    MUST_BE(sizeof(HASH_STATE) <= sizeof(EXPORT_HASH_STATE));
    // the following #define is used to move data from an aligned internal data
    // structure to a byte buffer (external format data.
#define CopyToOffset(value)                         \
    memcpy(&outBuf[offsetof(HASH_STATE, value)], \
           &internalFmt->value,                     \
           sizeof(internalFmt->value))
    // Copy the hashAlg
    CopyToOffset(hashAlg);
    CopyToOffset(type);
#ifdef HASH_STATE_SMAC
    if(internalFmt->type == HASH_STATE_SMAC)
    {
        memcpy(outBuf, internalFmt, sizeof(HASH_STATE));
        return;
    }
#endif
    if(internalFmt->type == HASH_STATE_HMAC)
    {
        HMAC_STATE* from = (HMAC_STATE*)internalFmt;
        memcpy(&outBuf[offsetof(HMAC_STATE, hmacKey)],
               &from->hmacKey,
               sizeof(from->hmacKey));
    }
    if(internalFmt->hashAlg != TPM_ALG_NULL)
        HASH_STATE_EXPORT(externalFmt, internalFmt);
}

//*** CryptHashImportState()
// This function is used to import the hash state. This function
// would be called to import a hash state when the context of a sequence object
// was being loaded.
void CryptHashImportState(
    PHASH_STATE internalFmt,         // OUT: the hash state formatted for use by
                                     //     the library
    PCEXPORT_HASH_STATE externalFmt  // IN: the exported hash state
)
{
    BYTE* inBuf = (BYTE*)externalFmt;
//
#define CopyFromOffset(value)                       \
    memcpy(&internalFmt->value,                     \
           &inBuf[offsetof(HASH_STATE, value)], \
           sizeof(internalFmt->value))

    // Copy the hashAlg of the byte-aligned input structure to the structure-aligned
    // internal structure.
    CopyFromOffset(hashAlg);
    CopyFromOffset(type);
    if(internalFmt->hashAlg != TPM_ALG_NULL)
    {
#ifdef HASH_STATE_SMAC
        if(internalFmt->type == HASH_STATE_SMAC)
        {
            memcpy(internalFmt, inBuf, sizeof(HASH_STATE));
            return;
        }
#endif
        internalFmt->def = CryptGetHashDef(internalFmt->hashAlg);
```

```
        HASH_STATE_IMPORT(internalFmt, inBuf);
        if(internalFmt->type == HASH_STATE_HMAC)
        {
            HMAC_STATE* to = (HMAC_STATE*)internalFmt;
            memcpy(&to->hmacKey,
                   &inBuf[offsetof(HMAC_STATE, hmacKey)],
                   sizeof(to->hmacKey));
        }
    }
}

//** State Modification Functions

//***HashEnd()
// Local function to complete a hash that uses the hashDef instead of an algorithm
// ID. This function is used to complete the hash and only return a partial digest.
// The return value is the size of the data copied.
static UINT16 HashEnd(PHASH_STATE hashState,  // IN: the hash state
                      UINT32      dOutSize,   // IN: the size of receive buffer
                      PBYTE       dOut        // OUT: the receive buffer
)
{
    BYTE temp[MAX_DIGEST_SIZE];
    if((hashState->hashAlg == TPM_ALG_NULL) || (hashState->type != HASH_STATE_HASH))
        dOutSize = 0;
    if(dOutSize > 0)
    {
        hashState->def = CryptGetHashDef(hashState->hashAlg);
        // Set the final size
        dOutSize = MIN(dOutSize, hashState->def->digestSize);
        // Complete into the temp buffer and then copy
        HASH_END(hashState, temp);
        // Don't want any other functions calling the HASH_END method
        // directly.
#undef HASH_END
        memcpy(dOut, &temp, dOutSize);
    }
    hashState->type = HASH_STATE_EMPTY;
    return (UINT16)dOutSize;
}

//*** CryptHashStart()
// Functions starts a hash stack
// Start a hash stack and returns the digest size. As a side effect, the
// value of 'stateSize' in hashState is updated to indicate the number of bytes
// of state that were saved. This function calls GetHashServer() and that function
// will put the TPM into failure mode if the hash algorithm is not supported.
//
// This function does not use the sequence parameter. If it is necessary to import
// or export context, this will start the sequence in a local state
// and export the state to the input buffer. Will need to add a flag to the state
// structure to indicate that it needs to be imported before it can be used.
// (BLEH).
//  Return Type: UINT16
//  0           hash is TPM_ALG_NULL
// >0           digest size
LIB_EXPORT UINT16 CryptHashStart(
    PHASH_STATE hashState,  // OUT: the running hash state
    TPM_ALG_ID  hashAlg     // IN: hash algorithm
)
{
    UINT16 retVal;

    TPM_DO_SELF_TEST(hashAlg);

    hashState->hashAlg = hashAlg;
```

```
    if(hashAlg == TPM_ALG_NULL)
    {
        retVal = 0;
    }
    else
    {
        hashState->def = CryptGetHashDef(hashAlg);
        HASH_START(hashState);
        retVal = hashState->def->digestSize;
    }
#undef HASH_START
    hashState->type = HASH_STATE_HASH;
    return retVal;
}

//*** CryptDigestUpdate()
// Add data to a hash or HMAC, SMAC stack.
//
void CryptDigestUpdate(PHASH_STATE hashState,  // IN: the hash context information
                       UINT32      dataSize,   // IN: the size of data to be added
                       const BYTE* data        // IN: data to be hashed
)
{
    if(hashState->hashAlg != TPM_ALG_NULL)
    {
        if((hashState->type == HASH_STATE_HASH)
           || (hashState->type == HASH_STATE_HMAC))
            HASH_DATA(hashState, dataSize, (BYTE*)data);
#if SMAC_IMPLEMENTED
        else if(hashState->type == HASH_STATE_SMAC)
            (hashState->state.smac.smacMethods.data)(
                &hashState->state.smac.state, dataSize, data);
#endif  // SMAC_IMPLEMENTED
        else
            FAIL(FATAL_ERROR_INTERNAL);
    }
    return;
}

//*** CryptHashEnd()
// Complete a hash or HMAC computation. This function will place the smaller of
// 'digestSize' or the size of the digest in 'dOut'. The number of bytes in the
// placed in the buffer is returned. If there is a failure, the returned value
// is <= 0.
//  Return Type: UINT16
//       0      no data returned
//      > 0     the number of bytes in the digest or dOutSize, whichever is smaller
LIB_EXPORT UINT16 CryptHashEnd(PHASH_STATE hashState,  // IN: the state of hash stack
                               UINT32      dOutSize,   // IN: size of digest buffer
                               BYTE*       dOut        // OUT: hash digest
)
{
    pAssert(hashState->type == HASH_STATE_HASH);
    return HashEnd(hashState, dOutSize, dOut);
}

//*** CryptHashBlock()
// Start a hash, hash a single block, update 'digest' and return the size of
// the results.
//
// The 'digestSize' parameter can be smaller than the digest. If so, only the more
// significant bytes are returned.
//  Return Type: UINT16
//  >= 0        number of bytes placed in 'dOut'
LIB_EXPORT UINT16 CryptHashBlock(TPM_ALG_ID  hashAlg,   // IN: The hash algorithm
                                 UINT32      dataSize,  // IN: size of buffer to hash
```

```
                                    const BYTE* data,       // IN: the buffer to hash
                                    UINT32 dOutSize,  // IN: size of the digest buffer
                                    BYTE*  dOut       // OUT: digest buffer
)
{
    HASH_STATE state;
    CryptHashStart(&state, hashAlg);
    CryptDigestUpdate(&state, dataSize, data);
    return HashEnd(&state, dOutSize, dOut);
}

//*** CryptDigestUpdate2B()
// This function updates a digest (hash or HMAC) with a TPM2B.
//
// This function can be used for both HMAC and hash functions so the
// 'digestState' is void so that either state type can be passed.
LIB_EXPORT void CryptDigestUpdate2B(PHASH_STATE  state,  // IN: the digest state
                                    const TPM2B* bIn     // IN: 2B containing the data
)
{
    // Only compute the digest if a pointer to the 2B is provided.
    // In CryptDigestUpdate(), if size is zero or buffer is NULL, then no change
    // to the digest occurs. This function should not provide a buffer if bIn is
    // not provided.
    pAssert(bIn != NULL);
    CryptDigestUpdate(state, bIn->size, bIn->buffer);
    return;
}

//*** CryptHashEnd2B()
// This function is the same as CryptCompleteHash() but the digest is
// placed in a TPM2B. This is the most common use and this is provided
// for specification clarity. 'digest.size' should be set to indicate the number of
// bytes to place in the buffer
//  Return Type: UINT16
//      >=0     the number of bytes placed in 'digest.buffer'
LIB_EXPORT UINT16 CryptHashEnd2B(
    PHASH_STATE state,  // IN: the hash state
    P2B         digest  // IN: the size of the buffer Out: requested
                        //     number of bytes
)
{
    return CryptHashEnd(state, digest->size, digest->buffer);
}

//*** CryptDigestUpdateInt()
// This function is used to include an integer value to a hash stack. The function
// marshals the integer into its canonical form before calling CryptDigestUpdate().
LIB_EXPORT void CryptDigestUpdateInt(
    void*  state,    // IN: the state of hash stack
    UINT32 intSize,  // IN: the size of 'intValue' in bytes
    UINT64 intValue  // IN: integer value to be hashed
)
{
#if LITTLE_ENDIAN_TPM
    intValue = REVERSE_ENDIAN_64(intValue);
#endif
    CryptDigestUpdate(state, intSize, &((BYTE*)&intValue)[8 - intSize]);
}

//** HMAC Functions

//*** CryptHmacStart()
// This function is used to start an HMAC using a temp
// hash context. The function does the initialization
// of the hash with the HMAC key XOR iPad and updates the
```

```
// HMAC key XOR oPad.
//
// The function returns the number of bytes in a digest produced by 'hashAlg'.
//  Return Type: UINT16
//  >= 0        number of bytes in digest produced by 'hashAlg' (may be zero)
//
LIB_EXPORT UINT16 CryptHmacStart(PHMAC_STATE state,    // IN/OUT: the state buffer
                                 TPM_ALG_ID  hashAlg,  // IN: the algorithm to use
                                 UINT16      keySize,  // IN: the size of the HMAC key
                                 const BYTE* key       // IN: the HMAC key
)
{
    PHASH_DEF hashDef;
    BYTE*     pb;
    UINT32    i;
    //
    hashDef = CryptGetHashDef(hashAlg);
    if(hashDef->digestSize != 0)
    {
        // If the HMAC key is larger than the hash block size, it has to be reduced
        // to fit. The reduction is a digest of the hashKey.
        if(keySize > hashDef->blockSize)
        {
            // if the key is too big, reduce it to a digest of itself
            state->hmacKey.t.size = CryptHashBlock(
                hashAlg, keySize, key, hashDef->digestSize, state->hmacKey.t.buffer);
        }
        else
        {
            memcpy(state->hmacKey.t.buffer, key, keySize);
            state->hmacKey.t.size = keySize;
        }
        // XOR the key with iPad (0x36)
        pb = state->hmacKey.t.buffer;
        for(i = state->hmacKey.t.size; i > 0; i--)
            *pb++ ^= 0x36;

        // if the keySize is smaller than a block, fill the rest with 0x36
        for(i = hashDef->blockSize - state->hmacKey.t.size; i > 0; i--)
            *pb++ = 0x36;

        // Increase the oPadSize to a full block
        state->hmacKey.t.size = hashDef->blockSize;

        // Start a new hash with the HMAC key
        // This will go in the caller's state structure and may be a sequence or not
        CryptHashStart((PHASH_STATE)state, hashAlg);
        CryptDigestUpdate(
            (PHASH_STATE)state, state->hmacKey.t.size, state->hmacKey.t.buffer);
        // XOR the key block with 0x5c ^ 0x36
        for(pb = state->hmacKey.t.buffer, i = hashDef->blockSize; i > 0; i--)
            *pb++ ^= (0x5c ^ 0x36);
    }
    // Set the hash algorithm
    state->hashState.hashAlg = hashAlg;
    // Set the hash state type
    state->hashState.type = HASH_STATE_HMAC;

    return hashDef->digestSize;
}

//*** CryptHmacEnd()
// This function is called to complete an HMAC. It will finish the current
// digest, and start a new digest. It will then add the oPadKey and the
// completed digest and return the results in dOut. It will not return more
// than dOutSize bytes.
```

```
//  Return Type: UINT16
//  >= 0        number of bytes in 'dOut' (may be zero)
LIB_EXPORT UINT16 CryptHmacEnd(PHMAC_STATE state,     // IN: the hash state buffer
                               UINT32      dOutSize,  // IN: size of digest buffer
                               BYTE*       dOut       // OUT: hash digest
)
{
    BYTE        temp[MAX_DIGEST_SIZE];
    PHASH_STATE hState = (PHASH_STATE)&state->hashState;

#if SMAC_IMPLEMENTED
    if(hState->type == HASH_STATE_SMAC)
        return (state->hashState.state.smac.smacMethods.end)(
            &state->hashState.state.smac.state, dOutSize, dOut);
#endif
    pAssert(hState->type == HASH_STATE_HMAC);
    hState->def = CryptGetHashDef(hState->hashAlg);
    // Change the state type for completion processing
    hState->type = HASH_STATE_HASH;
    if(hState->hashAlg == TPM_ALG_NULL)
        dOutSize = 0;
    else
    {
        // Complete the current hash
        HashEnd(hState, hState->def->digestSize, temp);
        // Do another hash starting with the oPad
        CryptHashStart(hState, hState->hashAlg);
        CryptDigestUpdate(hState, state->hmacKey.t.size, state->hmacKey.t.buffer);
        CryptDigestUpdate(hState, hState->def->digestSize, temp);
    }
    return HashEnd(hState, dOutSize, dOut);
}

//*** CryptHmacStart2B()
// This function starts an HMAC and returns the size of the digest
// that will be produced.
//
// This function is provided to support the most common use of starting an HMAC
// with a TPM2B key.
//
// The caller must provide a block of memory in which the hash sequence state
// is kept.  The caller should not alter the contents of this buffer until the
// hash sequence is completed or abandoned.
//
//  Return Type: UINT16
//      > 0     the digest size of the algorithm
//      = 0     the hashAlg was TPM_ALG_NULL
LIB_EXPORT UINT16 CryptHmacStart2B(
    PHMAC_STATE hmacState,  // OUT: the state of HMAC stack. It will be used
                            //      in HMAC update and completion
    TPMI_ALG_HASH hashAlg,  // IN: hash algorithm
    P2B           key       // IN: HMAC key
)
{
    return CryptHmacStart(hmacState, hashAlg, key->size, key->buffer);
}

//*** CryptHmacEnd2B()
//   This function is the same as CryptHmacEnd() but the HMAC result
//   is returned in a TPM2B which is the most common use.
//  Return Type: UINT16
//      >=0     the number of bytes placed in 'digest'
LIB_EXPORT UINT16 CryptHmacEnd2B(
    PHMAC_STATE hmacState,  // IN: the state of HMAC stack
    P2B         digest      // OUT: HMAC
)
```

```
{
    return CryptHmacEnd(hmacState, digest->size, digest->buffer);
}

//** Mask and Key Generation Functions
//*** CryptMGF_KDF()
// This function performs MGF1/KDF1 or KDF2 using the selected hash. KDF1 and KDF2 are
// T('n') = T('n'-1) || H('seed' || 'counter') with the difference being that, with
// KDF1, 'counter' starts at 0 but with KDF2, 'counter' starts at 1. The caller
// determines which version by setting the initial value of counter to either 0 or 1.
// Note: Any value that is not 0 is considered to be 1.
//
// This function returns the length of the mask produced which
// could be zero if the digest algorithm is not supported
//  Return Type: UINT16
//      0       hash algorithm was TPM_ALG_NULL
//    > 0       should be the same as 'mSize'
LIB_EXPORT UINT16 CryptMGF_KDF(UINT32 mSize,  // IN: length of the mask to be produced
                               BYTE*  mask,   // OUT: buffer to receive the mask
                               TPM_ALG_ID hashAlg,   // IN: hash to use
                               UINT32     seedSize,  // IN: size of the seed
                               BYTE*      seed,      // IN: seed size
                               UINT32     counter    // IN: counter initial value
)
{
    HASH_STATE hashState;
    PHASH_DEF  hDef = CryptGetHashDef(hashAlg);
    UINT32     hLen;
    UINT32     bytes;
    //
    // If there is no digest to compute return
    if((hDef->digestSize == 0) || (mSize == 0))
        return 0;
    if(counter != 0)
        counter = 1;
    hLen = hDef->digestSize;
    for(bytes = 0; bytes < mSize; bytes += hLen)
    {
        // Start the hash and include the seed and counter
        CryptHashStart(&hashState, hashAlg);
        CryptDigestUpdate(&hashState, seedSize, seed);
        CryptDigestUpdateInt(&hashState, 4, counter);
        // Get as much as will fit.
        CryptHashEnd(&hashState, MIN((mSize - bytes), hLen), &mask[bytes]);
        counter++;
    }
    return (UINT16)mSize;
}

//*** CryptKDFa()
// This function performs the key generation according to Part 1 of the
// TPM specification.
//
// This function returns the number of bytes generated which may be zero.
//
// The 'key' and 'keyStream' pointers are not allowed to be NULL. The other
// pointer values may be NULL. The value of 'sizeInBits' must be no larger
// than (2^18)-1 = 256K bits (32385 bytes).
//
// The 'once' parameter is set to allow incremental generation of a large
// value. If this flag is TRUE, 'sizeInBits' will be used in the HMAC computation
// but only one iteration of the KDF is performed. This would be used for
// XOR obfuscation so that the mask value can be generated in digest-sized
// chunks rather than having to be generated all at once in an arbitrarily
// large buffer and then XORed into the result. If 'once' is TRUE, then
// 'sizeInBits' must be a multiple of 8.
```

```
//
// Any error in the processing of this command is considered fatal.
//  Return Type: UINT16
//     0            hash algorithm is not supported or is TPM_ALG_NULL
//    > 0           the number of bytes in the 'keyStream' buffer
LIB_EXPORT UINT16 CryptKDFa(
    TPM_ALG_ID   hashAlg,       // IN: hash algorithm used in HMAC
    const TPM2B* key,           // IN: HMAC key
    const TPM2B* label,         // IN: a label for the KDF
    const TPM2B* contextU,      // IN: context U
    const TPM2B* contextV,      // IN: context V
    UINT32       sizeInBits,    // IN: size of generated key in bits
    BYTE*        keyStream,     // OUT: key buffer
    UINT32*      counterInOut,  // IN/OUT: caller may provide the iteration
                                //     counter for incremental operations to
                                //     avoid large intermediate buffers.
    UINT16 blocks               // IN: If non-zero, this is the maximum number
                                //     of blocks to be returned, regardless
                                //     of sizeInBits
)
{
    UINT32     counter = 0;  // counter value
    INT16      bytes;        // number of bytes to produce
    UINT16     generated;    // number of bytes generated
    BYTE*      stream = keyStream;
    HMAC_STATE hState;
    UINT16     digestSize = CryptHashGetDigestSize(hashAlg);

    pAssert(key != NULL && keyStream != NULL);

    TPM_DO_SELF_TEST(TPM_ALG_KDF1_SP800_108);

    if(digestSize == 0)
        return 0;

    if(counterInOut != NULL)
        counter = *counterInOut;

    // If the size of the request is larger than the numbers will handle,
    // it is a fatal error.
    pAssert(((sizeInBits + 7) / 8) <= INT16_MAX);

    // The number of bytes to be generated is the smaller of the sizeInBits bytes or
    // the number of requested blocks. The number of blocks is the smaller of the
    // number requested or the number allowed by sizeInBits. A partial block is
    // a full block.
    bytes = (blocks > 0) ? blocks * digestSize : (UINT16)BITS_TO_BYTES(sizeInBits);
    generated = bytes;

    // Generate required bytes
    for(; bytes > 0; bytes -= digestSize)
    {
        counter++;
        // Start HMAC
        if(CryptHmacStart(&hState, hashAlg, key->size, key->buffer) == 0)
            return 0;
        // Adding counter
        CryptDigestUpdateInt(&hState.hashState, 4, counter);

        // Adding label
        if(label != NULL)
            HASH_DATA(&hState.hashState, label->size, (BYTE*)label->buffer);
        // Add a null. SP108 is not very clear about when the 0 is needed but to
        // make this like the previous version that did not add an 0x00 after
        // a null-terminated string, this version will only add a null byte
        // if the label parameter did not end in a null byte, or if no label
```

```
        // is present.
        if((label == NULL) || (label->size == 0)
           || (label->buffer[label->size - 1] != 0))
            CryptDigestUpdateInt(&hState.hashState, 1, 0);
        // Adding contextU
        if(contextU != NULL)
            HASH_DATA(&hState.hashState, contextU->size, contextU->buffer);
        // Adding contextV
        if(contextV != NULL)
            HASH_DATA(&hState.hashState, contextV->size, contextV->buffer);
        // Adding size in bits
        CryptDigestUpdateInt(&hState.hashState, 4, sizeInBits);

        // Complete and put the data in the buffer
        CryptHmacEnd(&hState, bytes, stream);
        stream = &stream[digestSize];
    }
    // Masking in the KDF is disabled. If the calling function wants something
    // less than even number of bytes, then the caller should do the masking
    // because there is no universal way to do it here
    if(counterInOut != NULL)
        *counterInOut = counter;
    return generated;
}

//*** CryptKDFe()
// This function implements KDFe() as defined in TPM specification part 1.
//
// This function returns the number of bytes generated which may be zero.
//
// The 'Z' and 'keyStream' pointers are not allowed to be NULL. The other
// pointer values may be NULL. The value of 'sizeInBits' must be no larger
// than (2^18)-1 = 256K bits (32385 bytes).
// Any error in the processing of this command is considered fatal.
//  Return Type: UINT16
//     0            hash algorithm is not supported or is TPM_ALG_NULL
//    > 0           the number of bytes in the 'keyStream' buffer
//
LIB_EXPORT UINT16 CryptKDFe(TPM_ALG_ID   hashAlg,  // IN: hash algorithm used in HMAC
                            TPM2B*       Z,        // IN: Z
                            const TPM2B* label,    // IN: a label value for the KDF
                            TPM2B*       partyUInfo,  // IN: PartyUInfo
                            TPM2B*       partyVInfo,  // IN: PartyVInfo
                            UINT32 sizeInBits,  // IN: size of generated key in bits
                            BYTE*  keyStream    // OUT: key buffer
)
{
    HASH_STATE hashState;
    PHASH_DEF  hashDef = CryptGetHashDef(hashAlg);

    UINT32     counter = 0;  // counter value
    UINT16     hLen;
    BYTE*      stream = keyStream;
    INT16      bytes;  // number of bytes to generate

    pAssert(keyStream != NULL && Z != NULL && ((sizeInBits + 7) / 8) < INT16_MAX);
    //
    hLen  = hashDef->digestSize;
    bytes = (INT16)((sizeInBits + 7) / 8);
    if(hashAlg == TPM_ALG_NULL || bytes == 0)
        return 0;

    // Generate required bytes
    //The inner loop of that KDF uses:
    //  Hash[i] := H(counter | Z | OtherInfo) (5)
    // Where:
```

```
    //  Hash[i]         the hash generated on the i-th iteration of the loop.
    //  H()             an approved hash function
    //  counter         a 32-bit counter that is initialized to 1 and incremented
    //                  on each iteration
    //  Z               the X coordinate of the product of a public ECC key and a
    //                  different private ECC key.
    //  OtherInfo       a collection of qualifying data for the KDF defined below.
    //  In this specification, OtherInfo will be constructed by:
    //      OtherInfo := Use | PartyUInfo  | PartyVInfo
    for(; bytes > 0; stream = &stream[hLen], bytes = bytes - hLen)
    {
        if(bytes < hLen)
            hLen = bytes;
        counter++;
        // Do the hash
        CryptHashStart(&hashState, hashAlg);
        // Add counter
        CryptDigestUpdateInt(&hashState, 4, counter);

        // Add Z
        if(Z != NULL)
            CryptDigestUpdate2B(&hashState, Z);
        // Add label
        if(label != NULL)
            CryptDigestUpdate2B(&hashState, label);

        // NIST.SP.800-56Cr2.pdf section 4.1 states that no NULL
        // character is required here.
        // Note, this is different from KDFa which is specified in
        // NIST.SP.800-108r1.pdf section 4 (a NULL character is required
        // for that case).

        // Add PartyUInfo
        if(partyUInfo != NULL)
            CryptDigestUpdate2B(&hashState, partyUInfo);

        // Add PartyVInfo
        if(partyVInfo != NULL)
            CryptDigestUpdate2B(&hashState, partyVInfo);

        // Compute Hash. hLen was changed to be the smaller of bytes or hLen
        // at the start of each iteration.
        CryptHashEnd(&hashState, hLen, stream);
    }

    // Mask off bits if the required bits is not a multiple of byte size
    if((sizeInBits % 8) != 0)
        keyStream[0] &= ((1 << (sizeInBits % 8)) - 1);

    return (UINT16)((sizeInBits + 7) / 8);
}
```

7.144 /tpm/src/crypt/CryptPrime.c

```
//** Introduction
// This file contains the code for prime validation.

#include "Tpm.h"
#include "CryptPrime_fp.h"
#include "TpmMath_Util_fp.h"

//#define CPRI_PRIME
//#include "PrimeTable.h"

#include "CryptPrimeSieve_fp.h"
```

```
extern const uint32_t      s_LastPrimeInTable;
extern const uint32_t      s_PrimeTableSize;
extern const uint32_t      s_PrimesInTable;
extern const unsigned char s_PrimeTable[];
extern const Crypt_Int*    s_CompositeOfSmallPrimes;

//** Functions

//*** Root2()
// This finds ceil(sqrt(n)) to use as a stopping point for searching the prime
// table.
static uint32_t Root2(uint32_t n)
{
    int32_t last = (int32_t)(n >> 2);
    int32_t next = (int32_t)(n >> 1);
    int32_t diff;
    int32_t stop = 10;
    //
    // get a starting point
    for(; next != 0; last >>= 1, next >>= 2)
        ;
    last++;
    do
    {
        next = (last + (n / last)) >> 1;
        diff = next - last;
        last = next;
        if(stop-- == 0)
            FAIL(FATAL_ERROR_INTERNAL);
    } while(diff < -1 || diff > 1);
    if((n / next) > (unsigned)next)
        next++;
    pAssert(next != 0);
    pAssert(((n / next) <= (unsigned)next) && (n / (next + 1) < (unsigned)next));
    return next;
}

//*** IsPrimeInt()
// This will do a test of a word of up to 32-bits in size.
BOOL IsPrimeInt(uint32_t n)
{
    uint32_t i;
    uint32_t stop;
    if(n < 3 || ((n & 1) == 0))
        return (n == 2);
    if(n <= s_LastPrimeInTable)
    {
        n >>= 1;
        return ((s_PrimeTable[n >> 3] >> (n & 7)) & 1);
    }
    // Need to search
    stop = Root2(n) >> 1;
    // starting at 1 is equivalent to staring at  (1 << 1) + 1 = 3
    for(i = 1; i < stop; i++)
    {
        if((s_PrimeTable[i >> 3] >> (i & 7)) & 1)
            // see if this prime evenly divides the number
            if((n % ((i << 1) + 1)) == 0)
                return FALSE;
    }
    return TRUE;
}

//*** TpmMath_IsProbablyPrime()
// This function is used when the key sieve is not implemented. This function
```

```
// Will try to eliminate some of the obvious things before going on
// to perform MillerRabin as a final verification of primeness.
BOOL TpmMath_IsProbablyPrime(Crypt_Int*  prime,  // IN:
                             RAND_STATE* rand    // IN: the random state just
                             //     in case Miller-Rabin is required
)
{
    uint32_t leastSignificant32 = ExtMath_GetLeastSignificant32bits(prime);
    // is even?
    if((leastSignificant32 & 0x1) == 0)
        return FALSE;

    if(ExtMath_SizeInBits(prime) <= 32)
        return IsPrimeInt(leastSignificant32);

    // this s_LastPrimeInTable check guarantees that the full prime table check
    // is incorporated in IsPrimeInt.  If this fails then something like this
    // old code needs to be added back.
    // if(ExtMath_UnsignedCmpWord(prime, s_LastPrimeInTable) <= 0)
    // {
    //     // check fast prime table before doing slower checks
    //     crypt_uword_t temp = prime->d[0] >> 1;
    //     return ((s_PrimeTable[temp >> 3] >> (temp & 7)) & 1);
    // }
    MUST_BE(sizeof(s_LastPrimeInTable) <= 4);

    // check using GCD before doing a full Miller Rabin.
    {
        CRYPT_INT_VAR(gcd, LARGEST_NUMBER_BITS);
        ExtMath_GCD(gcd, prime, s_CompositeOfSmallPrimes);
        if(!ExtMath_IsEqualWord(gcd, 1))
            return FALSE;
    }
    return MillerRabin(prime, rand);
}

//*** MillerRabinRounds()
// Function returns the number of Miller-Rabin rounds necessary to give an
// error probability equal to the security strength of the prime. These values
// are from FIPS 186-3.
UINT32
MillerRabinRounds(UINT32 bits  // IN: Number of bits in the RSA prime
)
{
    if(bits < 511)
        return 8;  // don't really expect this
    if(bits < 1536)
        return 5;  // for 512 and 1K primes
    return 4;      // for 3K public modulus and greater
}

//*** MillerRabin()
// This function performs a Miller-Rabin test from FIPS 186-3. It does
// 'iterations' trials on the number. In all likelihood, if the number
// is not prime, the first test fails.
//  Return Type: BOOL
//      TRUE(1)         probably prime
//      FALSE(0)        composite
BOOL MillerRabin(Crypt_Int* bnW, RAND_STATE* rand)
{
    CRYPT_INT_MAX(bnWm1);
    CRYPT_PRIME_VAR(bnM);
    CRYPT_PRIME_VAR(bnB);
    CRYPT_PRIME_VAR(bnZ);
    BOOL         ret = FALSE;  // Assumed composite for easy exit
    unsigned int a;
```

```
    unsigned int j;
    int          wLen;
    int          i;
    int          iterations = MillerRabinRounds(ExtMath_SizeInBits(bnW));
    //
    INSTRUMENT_INC(MillerRabinTrials[PrimeIndex]);

    pAssert(bnW->size > 1);
    // Let a be the largest integer such that 2^a divides w1.
    ExtMath_SubtractWord(bnWm1, bnW, 1);
    pAssert(bnWm1->size != 0);

    // Since w is odd (w-1) is even so start at bit number 1 rather than 0
    // Get the number of bits in bnWm1 so that it doesn't have to be recomputed
    // on each iteration.
    i = (int)(bnWm1->size * RADIX_BITS);
    // Now find the largest power of 2 that divides w1
    for(a = 1; (a < (bnWm1->size * RADIX_BITS)) && (ExtMath_TestBit(bnWm1, a) == 0);
        a++)
    {
    }
    // 2. m = (w1) / 2^a
    ExtMath_ShiftRight(bnM, bnWm1, a);
    // 3. wlen = len (w).
    wLen = ExtMath_SizeInBits(bnW);
    // 4. For i = 1 to iterations do
    for(i = 0; i < iterations; i++)
    {
        // 4.1 Obtain a string b of wlen bits from an RBG.
        // Ensure that 1 < b < w1.
        // 4.2 If ((b <= 1) or (b >= w1)), then go to step 4.1.
        while(TpmMath_GetRandomInteger(bnB, wLen, rand)
              && ((ExtMath_UnsignedCmpWord(bnB, 1) <= 0)
                  || (ExtMath_UnsignedCmp(bnB, bnWm1) >= 0)))
            ;
        if(g_inFailureMode)
            return FALSE;

        // 4.3 z = b^m mod w.
        // if ModExp fails, then say this is not
        // prime and bail out.
        ExtMath_ModExp(bnZ, bnB, bnM, bnW);

        // 4.4 If ((z == 1) or (z = w == 1)), then go to step 4.7.
        if((ExtMath_UnsignedCmpWord(bnZ, 1) == 0)
           || (ExtMath_UnsignedCmp(bnZ, bnWm1) == 0))
            goto step4point7;
        // 4.5 For j = 1 to a  1 do.
        for(j = 1; j < a; j++)
        {
            // 4.5.1 z = z^2 mod w.
            ExtMath_ModMult(bnZ, bnZ, bnZ, bnW);
            // 4.5.2 If (z = w1), then go to step 4.7.
            if(ExtMath_UnsignedCmp(bnZ, bnWm1) == 0)
                goto step4point7;
            // 4.5.3 If (z = 1), then go to step 4.6.
            if(ExtMath_IsEqualWord(bnZ, 1))
                goto step4point6;
        }
        // 4.6 Return COMPOSITE.
step4point6:
        INSTRUMENT_INC(failedAtIteration[i]);
        goto end;
        // 4.7 Continue. Comment: Increment i for the do-loop in step 4.
step4point7:
        continue;
```

```
    }
    // 5. Return PROBABLY PRIME
    ret = TRUE;
end:
    return ret;
}

#if ALG_RSA

//*** RsaCheckPrime()
// This will check to see if a number is prime and appropriate for an
// RSA prime.
//
// This has different functionality based on whether we are using key
// sieving or not. If not, the number checked to see if it is divisible by
// the public exponent, then the number is adjusted either up or down
// in order to make it a better candidate. It is then checked for being
// probably prime.
//
// If sieving is used, the number is used to root a sieving process.
//
TPM_RC
RsaCheckPrime(Crypt_Int* prime, UINT32 exponent, RAND_STATE* rand)
{
#  if !RSA_KEY_SIEVE
    TPM_RC retVal = TPM_RC_SUCCESS;
    UINT32 modE   = ExtMath_ModWord(prime, exponent);

    NOT_REFERENCED(rand);

    if(modE == 0)
        // evenly divisible so add two keeping the number odd
        ExtMath_AddWord(prime, prime, 2);
    // want 0 != (p - 1) mod e
    // which is 1 != p mod e
    else if(modE == 1)
        // subtract 2 keeping number odd and insuring that
        // 0 != (p - 1) mod e
        ExtMath_SubtractWord(prime, prime, 2);

    if(TpmMath_IsProbablyPrime(prime, rand) == 0)
        ERROR_EXIT(g_inFailureMode ? TPM_RC_FAILURE : TPM_RC_VALUE);
Exit:
    return retVal;
#  else
    return PrimeSelectWithSieve(prime, exponent, rand);
#  endif
}

//*** RsaAdjustPrimeCandiate()
//
// For this math, we assume that the RSA numbers are fixed-point numbers with
// the decimal point to the "left" of the most significant bit. This approach helps
// make it clear what is happening with the MSb of the values.
// The two RSA primes have to be large enough so that their product will be a number
// with the necessary number of significant bits. For example, we want to be able
// to multiply two 1024-bit numbers to produce a number with 2028 significant bits. If
// we accept any 1024-bit prime that has its MSb set, then it is possible to produce a
// product that does not have the MSb SET. For example, if we use tiny keys of 16 bits
// and have two 8-bit 'primes' of 0x80, then the public key would be 0x4000 which is
// only 15-bits. So, what we need to do is made sure that each of the primes is large
// enough so that the product of the primes is twice as large as each prime. A little
// arithmetic will show that the only way to do this is to make sure that each of the
// primes is no less than root(2)/2. That's what this functions does.
// This function adjusts the candidate prime so that it is odd and >= root(2)/2.
// This allows the product of these two numbers to be .5, which, in fixed point
```

```
// notation means that the most significant bit is 1.
// For this routine, the root(2)/2 (0.7071067811865475) approximated with 0xB505
// which is, in fixed point, 0.7071075439453125 or an error of 0.000108%. Just setting
// the upper two bits would give a value > 0.75 which is an error of > 6%. Given the
// amount of time all the other computations take, reducing the error is not much of
// a cost, but it isn't totally required either.
//
// This function can be replaced with a function that just sets the two most
// significant bits of each prime candidate without introducing any computational
// issues.
//
static void RsaAdjustPrimeCandidate(BYTE* bigNumberBuffer, size_t bufSize)
{
    // first, ensure the last byte is odd, making the entire value odd
    bigNumberBuffer[bufSize - 1] |= 1;

    // second, get the most significant 32 bits.
    uint32_t msw = (bigNumberBuffer[0] << 24) | (bigNumberBuffer[1] << 16)
                 | (bigNumberBuffer[2] << 8) | (bigNumberBuffer[3] << 0);

    // Multiplying 0xff...f by 0x4AFB gives 0xff..f - 0xB5050...0
    uint32_t adjusted = (msw >> 16) * 0x4AFB;
    adjusted += ((msw & 0xFFFF) * 0x4AFB) >> 16;
    adjusted += 0xB5050000UL;

    // put the value back
    bigNumberBuffer[0] = (uint8_t)(adjusted >> 24);
    bigNumberBuffer[1] = (uint8_t)(adjusted >> 16);
    bigNumberBuffer[2] = (uint8_t)(adjusted >> 8);
    bigNumberBuffer[3] = (uint8_t)(adjusted >> 0);
}

//***TpmRsa_GeneratePrimeForRSA()
// Function to generate a prime of the desired size with the proper attributes
// for an RSA prime.
// succeeds, or enters failure mode.
TPM_RC TpmRsa_GeneratePrimeForRSA(
    Crypt_Int* prime,      // IN/OUT: points to the BN that will get the
                           //  random value
    UINT32      bits,      // IN: number of bits to get
    UINT32      exponent,  // IN: the exponent
    RAND_STATE* rand       // IN: the random state
)
{
    // Only try to handle specific sizes of keys.
    // this is necessary so the RsaAdjustPrimeCandidate function works correctly.
    pAssert((bits % 32) == 0);

    // create buffer large enough for the largest key
    TPM2B_TYPE(LARGEST, LARGEST_NUMBER);
    TPM2B_LARGEST large;

    NUMBYTES      bytes = (NUMBYTES)BITS_TO_BYTES(bits);
    BOOL          OK    = (bytes <= sizeof(large.t.buffer));
    BOOL          found = FALSE;
    while(OK && !found)
    {
        OK           = TpmMath_GetRandomBits(large.t.buffer, bits, rand);  // new
        large.t.size = bytes;
        RsaAdjustPrimeCandidate(large.t.buffer, bytes);
        // convert from 2B to Integer for prime checks
        OK = OK
             && (ExtMath_IntFromBytes(prime, large.t.buffer, large.t.size) != NULL);
        found = OK && (RsaCheckPrime(prime, exponent, rand) == TPM_RC_SUCCESS);
    }
```

```
    if(!OK)
    {
        FAIL(FATAL_ERROR_CRYPTO);
    }

    return (OK && found) ? TPM_RC_SUCCESS : TPM_RC_FAILURE;
}

#endif  // ALG_RSA
```

7.145 /tpm/src/crypt/CryptPrimeSieve.c

```
//** Includes and defines

#include "Tpm.h"

#if RSA_KEY_SIEVE

#  include "CryptPrimeSieve_fp.h"

// This determines the number of bits in the largest sieve field.
#  define MAX_FIELD_SIZE 2048

extern const uint32_t      s_LastPrimeInTable;
extern const uint32_t      s_PrimeTableSize;
extern const uint32_t      s_PrimesInTable;
extern const unsigned char s_PrimeTable[];

// This table is set of prime markers. Each entry is the prime value
// for the ((n + 1) * 1024) prime. That is, the entry in s_PrimeMarkers[1]
// is the value for the 2,048th prime. This is used in the PrimeSieve
// to adjust the limit for the prime search. When processing smaller
// prime candidates, fewer primes are checked directly before going to
// Miller-Rabin. As the prime grows, it is worth spending more time eliminating
// primes as, a) the density is lower, and b) the cost of Miller-Rabin is
// higher.
const uint32_t s_PrimeMarkersCount = 6;
const uint32_t s_PrimeMarkers[]    = {8167, 17881, 28183, 38891, 49871, 60961};
uint32_t       primeLimit;

//** Functions

//*** RsaAdjustPrimeLimit()
// This used during the sieve process. The iterator for getting the
// next prime (RsaNextPrime()) will return primes until it hits the
// limit (primeLimit) set up by this function. This causes the sieve
// process to stop when an appropriate number of primes have been
// sieved.
LIB_EXPORT void RsaAdjustPrimeLimit(uint32_t requestedPrimes)
{
    if(requestedPrimes == 0 || requestedPrimes > s_PrimesInTable)
        requestedPrimes = s_PrimesInTable;
    requestedPrimes = (requestedPrimes - 1) / 1024;
    if(requestedPrimes < s_PrimeMarkersCount)
        primeLimit = s_PrimeMarkers[requestedPrimes];
    else
        primeLimit = s_LastPrimeInTable;
    primeLimit >>= 1;
}

//*** RsaNextPrime()
// This the iterator used during the sieve process. The input is the
// last prime returned (or any starting point) and the output is the
// next higher prime. The function returns 0 when the primeLimit is
// reached.
```

```
LIB_EXPORT uint32_t RsaNextPrime(uint32_t lastPrime)
{
    if(lastPrime == 0)
        return 0;
    lastPrime >>= 1;
    for(lastPrime += 1; lastPrime <= primeLimit; lastPrime++)
    {
        if(((s_PrimeTable[lastPrime >> 3] >> (lastPrime & 0x7)) & 1) == 1)
            return ((lastPrime << 1) + 1);
    }
    return 0;
}

// This table contains a previously sieved table. It has
// the bits for 3, 5, and 7 removed. Because of the
// factors, it needs to be aligned to 105 and has
// a repeat of 105.
const BYTE seedValues[] = {0x16, 0x29, 0xcb, 0xa4, 0x65, 0xda, 0x30, 0x6c, 0x99, 0x96,
                           0x4c, 0x53, 0xa2, 0x2d, 0x52, 0x96, 0x49, 0xcb, 0xb4, 0x61,
                           0xd8, 0x32, 0x2d, 0x99, 0xa6, 0x44, 0x5b, 0xa4, 0x2c, 0x93,
                           0x96, 0x69, 0xc3, 0xb0, 0x65, 0x5a, 0x32, 0x4d, 0x89, 0xb6,
                           0x48, 0x59, 0x26, 0x2d, 0xd3, 0x86, 0x61, 0xcb, 0xb4, 0x64,
                           0x9a, 0x12, 0x6d, 0x91, 0xb2, 0x4c, 0x5a, 0xa6, 0x0d, 0xc3,
                           0x96, 0x69, 0xc9, 0x34, 0x25, 0xda, 0x22, 0x65, 0x99, 0xb4,
                           0x4c, 0x1b, 0x86, 0x2d, 0xd3, 0x92, 0x69, 0x4a, 0xb4, 0x45,
                           0xca, 0x32, 0x69, 0x99, 0x36, 0x0c, 0x5b, 0xa6, 0x25, 0xd3,
                           0x94, 0x68, 0x8b, 0x94, 0x65, 0xd2, 0x32, 0x6d, 0x18, 0xb6,
                           0x4c, 0x4b, 0xa6, 0x29, 0xd1};

#  define USE_NIBBLE

#  ifndef USE_NIBBLE
static const BYTE bitsInByte[256] =
    {0x00, 0x01, 0x01, 0x02, 0x01, 0x02, 0x02, 0x03, 0x01, 0x02, 0x02, 0x03, 0x02,
     0x03, 0x03, 0x04, 0x01, 0x02, 0x02, 0x03, 0x02, 0x03, 0x03, 0x04, 0x02, 0x03,
     0x03, 0x04, 0x03, 0x04, 0x04, 0x05, 0x01, 0x02, 0x02, 0x03, 0x02, 0x03, 0x03,
     0x04, 0x02, 0x03, 0x03, 0x04, 0x03, 0x04, 0x04, 0x05, 0x02, 0x03, 0x03, 0x04,
     0x03, 0x04, 0x04, 0x05, 0x03, 0x04, 0x04, 0x05, 0x04, 0x05, 0x05, 0x06, 0x01,
     0x02, 0x02, 0x03, 0x02, 0x03, 0x03, 0x04, 0x02, 0x03, 0x03, 0x04, 0x03, 0x04,
     0x04, 0x05, 0x02, 0x03, 0x03, 0x04, 0x03, 0x04, 0x04, 0x05, 0x03, 0x04, 0x04,
     0x05, 0x04, 0x05, 0x05, 0x06, 0x02, 0x03, 0x03, 0x04, 0x03, 0x04, 0x04, 0x05,
     0x03, 0x04, 0x04, 0x05, 0x04, 0x05, 0x05, 0x06, 0x03, 0x04, 0x04, 0x05, 0x04,
     0x05, 0x05, 0x06, 0x04, 0x05, 0x05, 0x06, 0x05, 0x06, 0x06, 0x07, 0x01, 0x02,
     0x02, 0x03, 0x02, 0x03, 0x03, 0x04, 0x02, 0x03, 0x03, 0x04, 0x03, 0x04, 0x04,
     0x05, 0x02, 0x03, 0x03, 0x04, 0x03, 0x04, 0x04, 0x05, 0x03, 0x04, 0x04, 0x05,
     0x04, 0x05, 0x05, 0x06, 0x02, 0x03, 0x03, 0x04, 0x03, 0x04, 0x04, 0x05, 0x03,
     0x04, 0x04, 0x05, 0x04, 0x05, 0x05, 0x06, 0x03, 0x04, 0x04, 0x05, 0x04, 0x05,
     0x05, 0x06, 0x04, 0x05, 0x05, 0x06, 0x05, 0x06, 0x06, 0x07, 0x02, 0x03, 0x03,
     0x04, 0x03, 0x04, 0x04, 0x05, 0x03, 0x04, 0x04, 0x05, 0x04, 0x05, 0x05, 0x06,
     0x03, 0x04, 0x04, 0x05, 0x04, 0x05, 0x05, 0x06, 0x04, 0x05, 0x05, 0x06, 0x05,
     0x06, 0x06, 0x07, 0x03, 0x04, 0x04, 0x05, 0x04, 0x05, 0x05, 0x06, 0x04, 0x05,
     0x05, 0x06, 0x05, 0x06, 0x06, 0x07, 0x04, 0x05, 0x05, 0x06, 0x05, 0x06, 0x06,
     0x07, 0x05, 0x06, 0x06, 0x07, 0x06, 0x07, 0x07, 0x08};
#    define BitsInByte(x) bitsInByte[(unsigned char)x]
#  else
const BYTE bitsInNibble[16] = {0x00,
                               0x01,
                               0x01,
                               0x02,
                               0x01,
                               0x02,
                               0x02,
                               0x03,
                               0x01,
                               0x02,
                               0x02,
```

```
                                0x03,
                                0x02,
                                0x03,
                                0x03,
                                0x04};
#    define BitsInByte(x)                          \
        (bitsInNibble[(unsigned char)(x) & 0xf] \
         + bitsInNibble[((unsigned char)(x) >> 4) & 0xf])
#  endif

//*** BitsInArry()
// This function counts the number of bits set in an array of bytes.
static int BitsInArray(const unsigned char* a,  // IN: A pointer to an array of bytes
                       unsigned int         aSize  // IN: the number of bytes to sum
)
{
    int j = 0;
    for(; aSize; a++, aSize--)
        j += BitsInByte(*a);
    return j;
}

//*** FindNthSetBit()
// This function finds the nth SET bit in a bit array. The 'n' parameter is
// between 1 and the number of bits in the array (always a multiple of 8).
// If called when the array does not have n bits set, it will return -1
//  Return Type: unsigned int
//      <0      no bit is set or no bit with the requested number is set
//      >=0    the number of the bit in the array that is the nth set
LIB_EXPORT int FindNthSetBit(
    const UINT16 aSize,  // IN: the size of the array to check
    const BYTE*  a,      // IN: the array to check
    const UINT32 n       // IN, the number of the SET bit
)
{
    UINT16 i;
    int    retValue;
    UINT32 sum = 0;
    BYTE   sel;

    //find the bit
    for(i = 0; (i < (int)aSize) && (sum < n); i++)
        sum += BitsInByte(a[i]);
    i--;
    // The chosen bit is in the byte that was just accessed
    // Compute the offset to the start of that byte
    retValue = i * 8 - 1;
    sel      = a[i];
    // Subtract the bits in the last byte added.
    sum -= BitsInByte(sel);
    // Now process the byte, one bit at a time.
    for(; (sel != 0) && (sum != n); retValue++, sel = sel >> 1)
        sum += (sel & 1) != 0;
    return (sum == n) ? retValue : -1;
}

typedef struct
{
    UINT32 prime;
    UINT16 count;
} SIEVE_MARKS;

// clang-format off
const SIEVE_MARKS sieveMarks[6] = {{31, 7},
                                   {73, 5},
                                   {241, 4},
```

```
                                    {1621, 3},
                                    {UINT16_MAX, 2},
                                    {UINT32_MAX, 1}};

const size_t MAX_SIEVE_MARKS = (sizeof(sieveMarks) / sizeof(sieveMarks[0]));
// clang-format on

//*** PrimeSieve()
// This function does a prime sieve over the input 'field' which has as its
// starting address the value in bnN. Since this initializes the Sieve
// using a precomputed field with the bits associated with 3, 5 and 7 already
// turned off, the value of pnN may need to be adjusted by a few counts to allow
// the precomputed field to be used without modification.
//
// To get better performance, one could address the issue of developing the
// composite numbers. When the size of the prime gets large, the time for doing
// the divisions goes up, noticeably. It could be better to develop larger composite
// numbers even if they need to be Crypt_Int*'s themselves. The object would be to
// reduce the number of times that the large prime is divided into a few large
// divides and then use smaller divides to get to the final 16 bit (or smaller)
// remainders.
LIB_EXPORT UINT32 PrimeSieve(Crypt_Int* bnN,    // IN/OUT: number to sieve
                             UINT32 fieldSize,  // IN: size of the field area in bytes
                             BYTE*  field       // IN: field
)
{
    UINT32 i;
    UINT32 j;
    UINT32 fieldBits = fieldSize * 8;
    UINT32 r;
    BYTE*  pField;
    INT32  iter;
    UINT32 adjust;
    UINT32 mark  = 0;
    UINT32 count = sieveMarks[0].count;
    UINT32 stop  = sieveMarks[0].prime;
    UINT32 composite;
    UINT32 pList[8];
    UINT32 next;

    pAssert(field != NULL && bnN != NULL);

    // If the remainder is odd, then subtracting the value will give an even number,
    // but we want an odd number, so subtract the 105+rem. Otherwise, just subtract
    // the even remainder.
    adjust = (UINT32)ExtMath_ModWord(bnN, 105);
    if(adjust & 1)
        adjust += 105;

    // Adjust the input number so that it points to the first number in a
    // aligned field.
    ExtMath_SubtractWord(bnN, bnN, adjust);
    //    pAssert(ExtMath_ModWord(bnN, 105) == 0);
    pField = field;
    for(i = fieldSize; i >= sizeof(seedValues);
        pField += sizeof(seedValues), i -= sizeof(seedValues))
    {
        memcpy(pField, seedValues, sizeof(seedValues));
    }
    if(i != 0)
        memcpy(pField, seedValues, i);

    // Cycle through the primes, clearing bits
    // Have already done 3, 5, and 7
    iter = 7;
```

```
#  define NEXT_PRIME(iter) (iter = RsaNextPrime(iter))
    // Get the next N primes where N is determined by the mark in the sieveMarks
    while((composite = NEXT_PRIME(iter)) != 0)
    {
        next       = 0;
        i          = count;
        pList[i--] = composite;
        for(; i > 0; i--)
        {
            next     = NEXT_PRIME(iter);
            pList[i] = next;
            if(next != 0)
                composite *= next;
        }
        // Get the remainder when dividing the base field address
        // by the composite
        composite = (UINT32)ExtMath_ModWord(bnN, composite);
        // 'composite' is divisible by the composite components. for each of the
        // composite components, divide 'composite'. That remainder (r) is used to
        // pick a starting point for clearing the array. The stride is equal to the
        // composite component. Note, the field only contains odd numbers. If the
        // field were expanded to contain all numbers, then half of the bits would
        // have already been cleared. We can save the trouble of clearing them a
        // second time by having a stride of 2*next. Or we can take all of the even
        // numbers out of the field and use a stride of 'next'
        for(i = count; i > 0; i--)
        {
            next = pList[i];
            if(next == 0)
                goto done;
            r = composite % next;
            // these computations deal with the fact that we have picked a field-sized
            // range that is aligned to a 105 count boundary. The problem is, this
field
            // only contains odd numbers. If we take our prime guess and walk through
all
            // the numbers using that prime as the 'stride', then every other 'stride'
is
            // going to be an even number. So, we are actually counting by 2 * the
stride
            // We want the count to start on an odd number at the start of our field.
That
            // is, we want to assume that we have counted up to the edge of the field
by
            // the 'stride' and now we are going to start flipping bits in the field
as we
            // continue to count up by 'stride'. If we take the base of our field and
            // divide by the stride, we find out how much we find out how short the
last
            // count was from reaching the edge of the bit field. Say we get a
quotient of
            // 3 and remainder of 1. This means that after 3 strides, we are 1 short
of
            // the start of the field and the next stride will either land within the
            // field or step completely over it. The confounding factor is that our
field
            // only contains odd numbers and our stride is actually 2 * stride. If the
            // quoitent is even, then that means that when we add 2 * stride, we are
going
            // to hit another even number. So, we have to know if we need to back off
            // by 1 stride before we start couting by 2 * stride.
            // We can tell from the remainder whether we are on an even or odd
            // stride when we hit the beginning of the table. If we are on an odd
stride
            // (r & 1), we would start half a stride in (next - r)/2. If we are on an
```

```
            // even stride, we need 0.5 strides (next - r/2) because the table only
has
            // odd numbers. If the remainder happens to be zero, then the start of the
            // table is on stride so no adjustment is necessary.
            if(r & 1)
                j = (next - r) / 2;
            else if(r == 0)
                j = 0;
            else
                j = next - (r / 2);
            for(; j < fieldBits; j += next)
                ClearBit(j, field, fieldSize);
        }
        if(next >= stop)
        {
            mark++;
            if(mark >= MAX_SIEVE_MARKS)
            {
                // prime iteration should have broken out of the loop before this.
                FAIL_EXIT(FATAL_ERROR_INTERNAL, i, 0);
            }
            count = sieveMarks[mark].count;
            stop  = sieveMarks[mark].prime;
        }
    }
done:
    i = BitsInArray(field, fieldSize);

Exit:
    INSTRUMENT_INC(totalFieldsSieved[PrimeIndex]);
    INSTRUMENT_ADD(bitsInFieldAfterSieve[PrimeIndex], i);
    INSTRUMENT_ADD(emptyFieldsSieved[PrimeIndex], (i == 0));
    return i;
}

#  ifdef SIEVE_DEBUG
static uint32_t fieldSize = 210;

//***SetFieldSize()
// Function to set the field size used for prime generation. Used for tuning.
LIB_EXPORT uint32_t SetFieldSize(uint32_t newFieldSize)
{
    if(newFieldSize == 0 || newFieldSize > MAX_FIELD_SIZE)
        fieldSize = MAX_FIELD_SIZE;
    else
        fieldSize = newFieldSize;
    return fieldSize;
}
#  endif  // SIEVE_DEBUG

//*** PrimeSelectWithSieve()
// This function will sieve the field around the input prime candidate. If the
// sieve field is not empty, one of the one bits in the field is chosen for testing
// with Miller-Rabin. If the value is prime, 'pnP' is updated with this value
// and the function returns success. If this value is not prime, another
// pseudo-random candidate is chosen and tested. This process repeats until
// all values in the field have been checked. If all bits in the field have
// been checked and none is prime, the function returns FALSE and a new random
// value needs to be chosen.
//  Return Type: TPM_RC
//      TPM_RC_FAILURE      TPM in failure mode, probably due to entropy source
//      TPM_RC_SUCCESS      candidate is probably prime
//      TPM_RC_NO_RESULT    candidate is not prime and couldn't find and alternative
//                          in the field
LIB_EXPORT TPM_RC PrimeSelectWithSieve(
    Crypt_Int*  candidate,  // IN/OUT: The candidate to filter
```

```
    UINT32      e,          // IN: the exponent
    RAND_STATE* rand        // IN: the random number generator state
)
{
    BYTE   field[MAX_FIELD_SIZE];
    UINT32 ones;
    INT32  chosen;
    CRYPT_PRIME_VAR(test);
    UINT32 modE;
#   ifndef SIEVE_DEBUG
    UINT32 fieldSize = MAX_FIELD_SIZE;
#   endif
    UINT32 primeSize;
    //
    // Adjust the field size and prime table list to fit the size of the prime
    // being tested. This is done to try to optimize the trade-off between the
    // dividing done for sieving and the time for Miller-Rabin. When the size
    // of the prime is large, the cost of Miller-Rabin is fairly high, as is the
    // cost of the sieving. However, the time for Miller-Rabin goes up considerably
    // faster than the cost of dividing by a number of primes.
    primeSize = ExtMath_SizeInBits(candidate);

    if(primeSize <= 512)
    {
        RsaAdjustPrimeLimit(1024);  // Use just the first 1024 primes
    }
    else if(primeSize <= 1024)
    {
        RsaAdjustPrimeLimit(4096);  // Use just the first 4K primes
    }
    else
    {
        RsaAdjustPrimeLimit(0);  // Use all available
    }

    // Save the low-order word to use as a search generator and make sure that
    // it has some interesting range to it
    uint32_t first = ExtMath_GetLeastSignificant32bits(candidate);
    first |= 0x80000000;

    // Sieve the field
    ones = PrimeSieve(candidate, fieldSize, field);

    // PrimeSieve shouldn't fail, but does call functions that may.
    if(!g_inFailureMode)
    {
        pAssert(ones > 0 && ones < (fieldSize * 8));
        for(; ones > 0; ones--)
        {
            // Decide which bit to look at and find its offset
            chosen = FindNthSetBit((UINT16)fieldSize, field, ((first % ones) + 1));

            if((chosen < 0) || (chosen >= (INT32)(fieldSize * 8)))
                FAIL(FATAL_ERROR_INTERNAL);

            // Set this as the trial prime
            ExtMath_AddWord(test, candidate, (crypt_uword_t)(chosen * 2));

            // The exponent might not have been one of the tested primes so
            // make sure that it isn't divisible and make sure that 0 != (p-1) mod e
            // Note: This is the same as 1 != p mod e
            modE = (UINT32)ExtMath_ModWord(test, e);
            if((modE != 0) && (modE != 1) && MillerRabin(test, rand))
            {
                ExtMath_Copy(candidate, test);
                return TPM_RC_SUCCESS;
```

```
            }
            // Clear the bit just tested
            ClearBit(chosen, field, fieldSize);
        }
        // Ran out of bits and couldn't find a prime in this field
        INSTRUMENT_INC(noPrimeFields[PrimeIndex]);
    }
    return (g_inFailureMode ? TPM_RC_FAILURE : TPM_RC_NO_RESULT);
}

#  if RSA_INSTRUMENT
static char a[256];

//*** PrintTuple()
char* PrintTuple(UINT32* i)
{
    sprintf(a, "{%d, %d, %d}", i[0], i[1], i[2]);
    return a;
}

#    define CLEAR_VALUE(x) memset(x, 0, sizeof(x))

//*** RsaSimulationEnd()
void RsaSimulationEnd(void)
{
    int    i;
    UINT32 averages[3];
    UINT32 nonFirst = 0;
    if((PrimeCounts[0] + PrimeCounts[1] + PrimeCounts[2]) != 0)
    {
        printf("Primes generated = %s\n", PrintTuple(PrimeCounts));
        printf("Fields sieved = %s\n", PrintTuple(totalFieldsSieved));
        printf("Fields with no primes = %s\n", PrintTuple(noPrimeFields));
        printf("Primes checked with Miller-Rabin = %s\n",
               PrintTuple(MillerRabinTrials));
        for(i = 0; i < 3; i++)
            averages[i] = (totalFieldsSieved[i] != 0
                           ? bitsInFieldAfterSieve[i] / totalFieldsSieved[i]
                           : 0);
        printf("Average candidates in field %s\n", PrintTuple(averages));
        for(i = 1; i < (sizeof(failedAtIteration) / sizeof(failedAtIteration[0]));
            i++)
            nonFirst += failedAtIteration[i];
        printf("Miller-Rabin failures not in first round = %d\n", nonFirst);
    }
    CLEAR_VALUE(PrimeCounts);
    CLEAR_VALUE(totalFieldsSieved);
    CLEAR_VALUE(noPrimeFields);
    CLEAR_VALUE(MillerRabinTrials);
    CLEAR_VALUE(bitsInFieldAfterSieve);
}

//*** GetSieveStats()
LIB_EXPORT void GetSieveStats(
    uint32_t* trials, uint32_t* emptyFields, uint32_t* averageBits)
{
    uint32_t totalBits;
    uint32_t fields;
    *trials      = MillerRabinTrials[0] + MillerRabinTrials[1] + MillerRabinTrials[2];
    *emptyFields = noPrimeFields[0] + noPrimeFields[1] + noPrimeFields[2];
    fields       = totalFieldsSieved[0] + totalFieldsSieved[1] + totalFieldsSieved[2];
    totalBits    = bitsInFieldAfterSieve[0] + bitsInFieldAfterSieve[1]
                 + bitsInFieldAfterSieve[2];
    if(fields != 0)
        *averageBits = totalBits / fields;
    else
```

```
        *averageBits = 0;
    CLEAR_VALUE(PrimeCounts);
    CLEAR_VALUE(totalFieldsSieved);
    CLEAR_VALUE(noPrimeFields);
    CLEAR_VALUE(MillerRabinTrials);
    CLEAR_VALUE(bitsInFieldAfterSieve);
}
#  endif

#endif  // RSA_KEY_SIEVE

#if !RSA_INSTRUMENT

//*** RsaSimulationEnd()
// Stub for call when not doing instrumentation.
void RsaSimulationEnd(void)
{
    return;
}
#endif
```

7.146 /tpm/src/crypt/CryptRand.c

```
//** Introduction
// This file implements a DRBG with a behavior according to SP800-90A using
// a block cypher. This is also compliant to ISO/IEC 18031:2011(E) C.3.2.
//
// A state structure is created for use by TPM.lib and functions
// within the CryptoEngine my use their own state structures when they need to have
// deterministic values.
//
// A debug mode is available that allows the random numbers generated for TPM.lib
// to be repeated during runs of the simulator. The switch for it is in
// TpmBuildSwitches.h. It is USE_DEBUG_RNG.
//
//
// This is the implementation layer of CTR DRGB mechanism as defined in SP800-90A
// and the functions are organized as closely as practical to the organization in
// SP800-90A. It is intended to be compiled as a separate module that is linked
// with a secure application so that both reside inside the same boundary
// [SP 800-90A 8.5]. The secure application in particular manages the accesses
// protected storage for the state of the DRBG instantiations, and supplies the
// implementation functions here with a valid pointer to the working state of the
// given instantiations (as a DRBG_STATE structure).
//
// This DRBG mechanism implementation does not support prediction resistance. Thus
// 'prediction_resistance_flag' is omitted from Instantiate_function(),
// Reseed_function(), Generate_function() argument lists [SP 800-90A 9.1, 9.2,
// 9.3], as well as from the working state data structure DRBG_STATE [SP 800-90A
// 9.1].
//
// This DRBG mechanism implementation always uses the highest security strength of
// available in the block ciphers. Thus 'requested_security_strength' parameter is
// omitted from Instantiate_function() and Generate_function() argument lists
// [SP 800-90A 9.1, 9.2, 9.3], as well as from the working state data structure
// DRBG_STATE [SP 800-90A 9.1].
//
// Internal functions (ones without Crypt prefix) expect validated arguments and
// therefore use assertions instead of runtime parameter checks and mostly return
// void instead of a status value.

#include "Tpm.h"

// Pull in the test vector definitions and define the space
#include "PRNG_TestVectors.h"
```

```
const BYTE DRBG_NistTestVector_Entropy[]         = {DRBG_TEST_INITIATE_ENTROPY};
const BYTE DRBG_NistTestVector_GeneratedInterm[] = {DRBG_TEST_GENERATED_INTERM};

const BYTE DRBG_NistTestVector_EntropyReseed[]   = {DRBG_TEST_RESEED_ENTROPY};
const BYTE DRBG_NistTestVector_Generated[]       = {DRBG_TEST_GENERATED};

//** Derivation Functions
//*** Description
// The functions in this section are used to reduce the personalization input values
// to make them usable as input for reseeding and instantiation. The overall
// behavior is intended to produce the same results as described in SP800-90A,
// section 10.4.2 "Derivation Function Using a Block Cipher Algorithm
// (Block_Cipher_df)." The code is broken into several subroutines to deal with the
// fact that the data used for personalization may come in several separate blocks
// such as a Template hash and a proof value and a primary seed.

//*** Derivation Function Defines and Structures

#define DF_COUNT (DRBG_KEY_SIZE_WORDS / DRBG_IV_SIZE_WORDS + 1)
#if DRBG_KEY_SIZE_BITS != 128 && DRBG_KEY_SIZE_BITS != 256
#  error "CryptRand.c only written for AES with 128- or 256-bit keys."
#endif

typedef tpmKeyScheduleAES DRBG_KEY_SCHEDULE;

typedef struct
{
    DRBG_KEY_SCHEDULE keySchedule;
    DRBG_IV           iv[DF_COUNT];
    DRBG_IV           out1;
    DRBG_IV           buf;
    int               contents;
} DF_STATE, *PDF_STATE;

//*** DfCompute()
// This function does the incremental update of the derivation function state. It
// encrypts the 'iv' value and XOR's the results into each of the blocks of the
// output. This is equivalent to processing all of input data for each output block.
static void DfCompute(PDF_STATE dfState)
{
    int            i;
    int            iv;
    crypt_uword_t* pIv;
    crypt_uword_t  temp[DRBG_IV_SIZE_WORDS] = {0};
    //
    for(iv = 0; iv < DF_COUNT; iv++)
    {
        pIv = (crypt_uword_t*)&dfState->iv[iv].words[0];
        for(i = 0; i < DRBG_IV_SIZE_WORDS; i++)
        {
            temp[i] ^= pIv[i] ^ dfState->buf.words[i];
        }
        DRBG_ENCRYPT(&dfState->keySchedule, &temp, pIv);
    }
    for(i = 0; i < DRBG_IV_SIZE_WORDS; i++)
        dfState->buf.words[i] = 0;
    dfState->contents = 0;
}

//*** DfStart()
// This initializes the output blocks with an encrypted counter value and
// initializes the key schedule.
static void DfStart(PDF_STATE dfState, uint32_t inputLength)
{
    BYTE       init[8];
```

```
    int         i;
    UINT32      drbgSeedSize = sizeof(DRBG_SEED);

    const BYTE dfKey[DRBG_KEY_SIZE_BYTES] =
    { 0x00,
      0x01,
      0x02,
      0x03,
      0x04,
      0x05,
      0x06,
      0x07,
      0x08,
      0x09,
      0x0a,
      0x0b,
      0x0c,
      0x0d,
      0x0e,
      0x0f
#if DRBG_KEY_SIZE_BYTES > 16
      ,
      0x10,
      0x11,
      0x12,
      0x13,
      0x14,
      0x15,
      0x16,
      0x17,
      0x18,
      0x19,
      0x1a,
      0x1b,
      0x1c,
      0x1d,
      0x1e,
      0x1f
#endif
    };
    memset(dfState, 0, sizeof(DF_STATE));
    DRBG_ENCRYPT_SETUP(&dfKey[0], DRBG_KEY_SIZE_BITS, &dfState->keySchedule);
    // Create the first chaining values
    for(i = 0; i < DF_COUNT; i++)
        ((BYTE*)&dfState->iv[i])[3] = (BYTE)i;
    DfCompute(dfState);
    // initialize the first 64 bits of the IV in a way that doesn't depend
    // on the size of the words used.
    UINT32_TO_BYTE_ARRAY(inputLength, init);
    UINT32_TO_BYTE_ARRAY(drbgSeedSize, &init[4]);
    memcpy(&dfState->iv[0], init, 8);
    dfState->contents = 4;
}

//*** DfUpdate()
// This updates the state with the input data. A byte at a time is moved into the
// state buffer until it is full and then that block is encrypted by DfCompute().
static void DfUpdate(PDF_STATE dfState, int size, const BYTE* data)
{
    while(size > 0)
    {
        int toFill = DRBG_IV_SIZE_BYTES - dfState->contents;
        if(size < toFill)
            toFill = size;
        // Copy as many bytes as there are or until the state buffer is full
        memcpy(&dfState->buf.bytes[dfState->contents], data, toFill);
```

```
        // Reduce the size left by the amount copied
        size -= toFill;
        // Advance the data pointer by the amount copied
        data += toFill;
        // increase the buffer contents count by the amount copied
        dfState->contents += toFill;
        pAssert(dfState->contents <= DRBG_IV_SIZE_BYTES);
        // If we have a full buffer, do a computation pass.
        if(dfState->contents == DRBG_IV_SIZE_BYTES)
            DfCompute(dfState);
    }
}

//*** DfEnd()
// This function is called to get the result of the derivation function computation.
// If the buffer is not full, it is padded with zeros. The output buffer is
// structured to be the same as a DRBG_SEED value so that the function can return
// a pointer to the DRBG_SEED value in the DF_STATE structure.
static DRBG_SEED* DfEnd(PDF_STATE dfState)
{
    // Since DfCompute is always called when a buffer is full, there is always
    // space in the buffer for the terminator
    dfState->buf.bytes[dfState->contents++] = 0x80;
    // If the buffer is not full, pad with zeros
    while(dfState->contents < DRBG_IV_SIZE_BYTES)
        dfState->buf.bytes[dfState->contents++] = 0;
    // Do a final state update
    DfCompute(dfState);
    return (DRBG_SEED*)&dfState->iv;
}

//*** DfBuffer()
// Function to take an input buffer and do the derivation function to produce a
// DRBG_SEED value that can be used in DRBG_Reseed();
static DRBG_SEED* DfBuffer(DRBG_SEED* output,  // OUT: receives the result
                           int        size,    // IN: size of the buffer to add
                           BYTE*      buf      // IN: address of the buffer
)
{
    DF_STATE dfState;
    if(size == 0 || buf == NULL)
        return NULL;
    // Initialize the derivation function
    DfStart(&dfState, size);
    DfUpdate(&dfState, size, buf);
    DfEnd(&dfState);
    memcpy(output, &dfState.iv[0], sizeof(DRBG_SEED));
    return output;
}

//*** DRBG_GetEntropy()
// Even though this implementation never fails, it may get blocked
// indefinitely long in the call to get entropy from the platform
// (DRBG_GetEntropy32()).
// This function is only used during instantiation of the DRBG for
// manufacturing and on each start-up after an non-orderly shutdown.
//
//  Return Type: BOOL
//      TRUE(1)         requested entropy returned
//      FALSE(0)        entropy Failure
BOOL DRBG_GetEntropy(UINT32 requiredEntropy,  // IN: requested number of bytes of full
                                              //     entropy
                     BYTE* entropy  // OUT: buffer to return collected entropy
)
{
#if !USE_DEBUG_RNG
```

```
    UINT32 obtainedEntropy;
    INT32  returnedEntropy;

    // If in debug mode, always use the self-test values for initialization
    if(IsSelfTest())
    {
#endif
        // If doing simulated DRBG, then check to see if the
        // entropyFailure condition is being tested
        if(!IsEntropyBad())
        {
            // In self-test, the caller should be asking for exactly the seed
            // size of entropy.
            pAssert(requiredEntropy == sizeof(DRBG_NistTestVector_Entropy));
            memcpy(entropy,
                   DRBG_NistTestVector_Entropy,
                   sizeof(DRBG_NistTestVector_Entropy));
        }
#if !USE_DEBUG_RNG
    }
    else if(!IsEntropyBad())
    {
        // Collect entropy
        // Note: In debug mode, the only "entropy" value ever returned
        // is the value of the self-test vector.
        for(returnedEntropy = 1, obtainedEntropy = 0;
            obtainedEntropy < requiredEntropy && !IsEntropyBad();
            obtainedEntropy += returnedEntropy)
        {
            returnedEntropy = _plat__GetEntropy(&entropy[obtainedEntropy],
                                                requiredEntropy - obtainedEntropy);
            if(returnedEntropy <= 0)
                SetEntropyBad();
        }
    }
#endif
    return !IsEntropyBad();
}

//*** IncrementIv()
// This function increments the IV value by 1. It is used by EncryptDRBG().
void IncrementIv(DRBG_IV* iv)
{
    BYTE* ivP = ((BYTE*)iv) + DRBG_IV_SIZE_BYTES;
    while((--ivP >= (BYTE*)iv) && ((*ivP = ((*ivP + 1) & 0xFF)) == 0))
        ;
}

//*** EncryptDRBG()
// This does the encryption operation for the DRBG. It will encrypt
// the input state counter (IV) using the state key. Into the output
// buffer for as many times as it takes to generate the required
// number of bytes.
static BOOL EncryptDRBG(BYTE*              dOut,
                        UINT32             dOutBytes,
                        DRBG_KEY_SCHEDULE* keySchedule,
                        DRBG_IV*           iv,
                        UINT32* lastValue  // Points to the last output value
)
{
#if FIPS_COMPLIANT
    // For FIPS compliance, the DRBG has to do a continuous self-test to make sure
that
    // no two consecutive values are the same. This overhead is not incurred if the
TPM
```

```
    // is not required to be FIPS compliant
    //
    UINT32 temp[DRBG_IV_SIZE_BYTES / sizeof(UINT32)];
    int    i;
    BYTE*  p;

    for(; dOutBytes > 0;)
    {
        // Increment the IV before each encryption (this is what makes this
        // different from normal counter-mode encryption
        IncrementIv(iv);
        DRBG_ENCRYPT(keySchedule, iv, temp);
// Expect a 16 byte block
#  if DRBG_IV_SIZE_BITS != 128
#    error "Unsuppored IV size in DRBG"
#  endif
        if((lastValue[0] == temp[0]) && (lastValue[1] == temp[1])
           && (lastValue[2] == temp[2]) && (lastValue[3] == temp[3]))
        {
            FAIL_BOOL(FATAL_ERROR_ENTROPY);
        }
        lastValue[0] = temp[0];
        lastValue[1] = temp[1];
        lastValue[2] = temp[2];
        lastValue[3] = temp[3];
        i            = MIN(dOutBytes, DRBG_IV_SIZE_BYTES);
        dOutBytes -= i;
        for(p = (BYTE*)temp; i > 0; i--)
            *dOut++ = *p++;
    }
#else  // version without continuous self-test
    NOT_REFERENCED(lastValue);
    for(; dOutBytes >= DRBG_IV_SIZE_BYTES;
        dOut = &dOut[DRBG_IV_SIZE_BYTES], dOutBytes -= DRBG_IV_SIZE_BYTES)
    {
        // Increment the IV
        IncrementIv(iv);
        DRBG_ENCRYPT(keySchedule, iv, dOut);
    }
    // If there is a partial, generate into a block-sized
    // temp buffer and copy to the output.
    if(dOutBytes != 0)
    {
        BYTE temp[DRBG_IV_SIZE_BYTES];
        // Increment the IV
        IncrementIv(iv);
        DRBG_ENCRYPT(keySchedule, iv, temp);
        memcpy(dOut, temp, dOutBytes);
    }
#endif
    return TRUE;
}

//*** DRBG_Update()
// This function performs the state update function.
// According to SP800-90A, a temp value is created by doing CTR mode
// encryption of 'providedData' and replacing the key and IV with
// these values. The one difference is that, with counter mode, the
// IV is incremented after each block is encrypted and in this
// operation, the counter is incremented before each block is
// encrypted. This function implements an 'optimized' version
// of the algorithm in that it does the update of the drbgState->seed
// in place and then 'providedData' is XORed into drbgState->seed
// to complete the encryption of 'providedData'. This works because
// the IV is the last thing that gets encrypted.
//
```

```
static BOOL DRBG_Update(
    DRBG_STATE*         drbgState,    // IN:OUT state to update
    DRBG_KEY_SCHEDULE* keySchedule,  // IN: the key schedule (optional)
    DRBG_SEED*          providedData  // IN: additional data
)
{
    UINT32             i;
    BYTE*              temp = (BYTE*)&drbgState->seed;
    DRBG_KEY*          key  = pDRBG_KEY(&drbgState->seed);
    DRBG_IV*           iv   = pDRBG_IV(&drbgState->seed);
    DRBG_KEY_SCHEDULE localKeySchedule;
    //
    pAssert(drbgState->magic == DRBG_MAGIC);

    // If an key schedule was not provided, make one
    if(keySchedule == NULL)
    {
        if(DRBG_ENCRYPT_SETUP((BYTE*)key, DRBG_KEY_SIZE_BITS, &localKeySchedule) != 0)
        {
            FAIL_BOOL(FATAL_ERROR_INTERNAL);
        }
        keySchedule = &localKeySchedule;
    }
    // Encrypt the temp value

    EncryptDRBG(temp, sizeof(DRBG_SEED), keySchedule, iv, drbgState->lastValue);
    if(providedData != NULL)
    {
        BYTE* pP = (BYTE*)providedData;
        for(i = DRBG_SEED_SIZE_BYTES; i != 0; i--)
            *temp++ ^= *pP++;
    }
    // Since temp points to the input key and IV, we are done and
    // don't need to copy the resulting 'temp' to drbgState->seed
    return TRUE;
}

//*** DRBG_Reseed()
// This function is used when reseeding of the DRBG is required. If
// entropy is provided, it is used in lieu of using hardware entropy.
// Note: the provided entropy must be the required size.
//
//  Return Type: BOOL
//      TRUE(1)         reseed succeeded
//      FALSE(0)        reseed failed, probably due to the entropy generation
BOOL DRBG_Reseed(DRBG_STATE* drbgState,        // IN: the state to update
                 DRBG_SEED*  providedEntropy,  // IN: entropy
                 DRBG_SEED*  additionalData    // IN:
)
{
    DRBG_SEED seed;

    pAssert((drbgState != NULL) && (drbgState->magic == DRBG_MAGIC));

    if(providedEntropy == NULL)
    {
        providedEntropy = &seed;
        if(!DRBG_GetEntropy(sizeof(DRBG_SEED), (BYTE*)providedEntropy))
            return FALSE;
    }
    if(additionalData != NULL)
    {
        unsigned int i;

        // XOR the provided data into the provided entropy
        for(i = 0; i < sizeof(DRBG_SEED); i++)
```

```
            ((BYTE*)providedEntropy)[i] ^= ((BYTE*)additionalData)[i];
    }
    DRBG_Update(drbgState, NULL, providedEntropy);

    drbgState->reseedCounter = 1;

    return TRUE;
}

//*** DRBG_SelfTest()
// This is run when the DRBG is instantiated and at startup.
//
//  Return Type: BOOL
//      TRUE(1)         test OK
//      FALSE(0)        test failed
BOOL DRBG_SelfTest(void)
{
    BYTE       buf[sizeof(DRBG_NistTestVector_Generated)];
    DRBG_SEED  seed;
    UINT32     i;
    BYTE*      p;
    DRBG_STATE testState;
    //
    pAssert(!IsSelfTest());

    SetSelfTest();
    SetDrbgTested();
    // Do an instantiate
    if(!DRBG_Instantiate(&testState, 0, NULL))
        return FALSE;
#if DRBG_DEBUG_PRINT
    dbgDumpMemBlock(
        pDRBG_KEY(&testState), DRBG_KEY_SIZE_BYTES, "Key after Instantiate");
    dbgDumpMemBlock(
        pDRBG_IV(&testState), DRBG_IV_SIZE_BYTES, "Value after Instantiate");
#endif
    if(DRBG_Generate((RAND_STATE*)&testState, buf, sizeof(buf)) == 0)
        return FALSE;
#if DRBG_DEBUG_PRINT
    dbgDumpMemBlock(
        pDRBG_KEY(&testState.seed), DRBG_KEY_SIZE_BYTES, "Key after 1st Generate");
    dbgDumpMemBlock(
        pDRBG_IV(&testState.seed), DRBG_IV_SIZE_BYTES, "Value after 1st Generate");
#endif
    if(memcmp(buf, DRBG_NistTestVector_GeneratedInterm, sizeof(buf)) != 0)
        return FALSE;
    memcpy(seed.bytes, DRBG_NistTestVector_EntropyReseed, sizeof(seed));
    DRBG_Reseed(&testState, &seed, NULL);
#if DRBG_DEBUG_PRINT
    dbgDumpMemBlock((BYTE*)pDRBG_KEY(&testState.seed),
                    DRBG_KEY_SIZE_BYTES,
                    "Key after 2nd Generate");
    dbgDumpMemBlock((BYTE*)pDRBG_IV(&testState.seed),
                    DRBG_IV_SIZE_BYTES,
                    "Value after 2nd Generate");
    dbgDumpMemBlock(buf, sizeof(buf), "2nd Generated");
#endif
    if(DRBG_Generate((RAND_STATE*)&testState, buf, sizeof(buf)) == 0)
        return FALSE;
    if(memcmp(buf, DRBG_NistTestVector_Generated, sizeof(buf)) != 0)
        return FALSE;
    ClearSelfTest();

    DRBG_Uninstantiate(&testState);
    for(p = (BYTE*)&testState, i = 0; i < sizeof(DRBG_STATE); i++)
    {
```

```
        if(*p++)
            return FALSE;
    }
    // Simulate hardware failure to make sure that we get an error when
    // trying to instantiate
    SetEntropyBad();
    if(DRBG_Instantiate(&testState, 0, NULL))
        return FALSE;
    ClearEntropyBad();

    return TRUE;
}

//** Public Interface
//*** Description
// The functions in this section are the interface to the RNG. These
// are the functions that are used by TPM.lib.

//*** CryptRandomStir()
// This function is used to cause a reseed. A DRBG_SEED amount of entropy is
// collected from the hardware and then additional data is added.
//
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        failure of the entropy generator
LIB_EXPORT TPM_RC CryptRandomStir(UINT16 additionalDataSize, BYTE* additionalData)
{
#if !USE_DEBUG_RNG
    DRBG_SEED tmpBuf;
    DRBG_SEED dfResult;
    //
    // All reseed with outside data starts with a buffer full of entropy
    if(!DRBG_GetEntropy(sizeof(tmpBuf), (BYTE*)&tmpBuf))
        return TPM_RC_NO_RESULT;

    DRBG_Reseed(&drbgDefault,
                &tmpBuf,
                DfBuffer(&dfResult, additionalDataSize, additionalData));
    drbgDefault.reseedCounter = 1;

    return TPM_RC_SUCCESS;

#else
    // If doing debug, use the input data as the initial setting for the RNG state
    // so that the test can be reset at any time.
    // Note: If this is called with a data size of 0 or less, nothing happens. The
    // presumption is that, in a debug environment, the caller will have specific
    // values for initialization, so this check is just a simple way to prevent
    // inadvertent programming errors from screwing things up. This doesn't use an
    // pAssert() because the non-debug version of this function will accept these
    // parameters as meaning that there is no additionalData and only hardware
    // entropy is used.
    if((additionalDataSize > 0) && (additionalData != NULL))
    {
        memset(drbgDefault.seed.bytes, 0, sizeof(drbgDefault.seed.bytes));
        memcpy(drbgDefault.seed.bytes,
               additionalData,
               MIN(additionalDataSize, sizeof(drbgDefault.seed.bytes)));
    }
    drbgDefault.reseedCounter = 1;

    return TPM_RC_SUCCESS;
#endif
}

//*** CryptRandomGenerate()
// Generate a 'randomSize' number or random bytes.
```

```
LIB_EXPORT UINT16 CryptRandomGenerate(UINT16 randomSize, BYTE* buffer)
{
    return DRBG_Generate((RAND_STATE*)&drbgDefault, buffer, randomSize);
}

//*** DRBG_InstantiateSeededKdf()
// This function is used to instantiate a KDF-based RNG. This is used for derivations.
// This function always returns TRUE.
LIB_EXPORT BOOL DRBG_InstantiateSeededKdf(
    KDF_STATE*   state,    // OUT: buffer to hold the state
    TPM_ALG_ID   hashAlg,  // IN: hash algorithm
    TPM_ALG_ID   kdf,      // IN: the KDF to use
    TPM2B*       seed,     // IN: the seed to use
    const TPM2B* label,    // IN: a label for the generation process.
    TPM2B*       context,  // IN: the context value
    UINT32       limit     // IN: Maximum number of bits from the KDF
)
{
    state->magic           = KDF_MAGIC;
    state->limit           = limit;
    state->seed            = seed;
    state->hash            = hashAlg;
    state->kdf             = kdf;
    state->label           = label;
    state->context         = context;
    state->digestSize      = CryptHashGetDigestSize(hashAlg);
    state->counter         = 0;
    state->residual.t.size = 0;
    return TRUE;
}

//*** DRBG_AdditionalData()
// Function to reseed the DRBG with additional entropy. This is normally called
// before computing the protection value of a primary key in the Endorsement
// hierarchy.
LIB_EXPORT void DRBG_AdditionalData(DRBG_STATE* drbgState,  // IN:OUT state to update
                                    TPM2B* additionalData  // IN: value to incorporate
)
{
    DRBG_SEED dfResult;
    if(drbgState->magic == DRBG_MAGIC)
    {
        DfBuffer(&dfResult, additionalData->size, additionalData->buffer);
        DRBG_Reseed(drbgState, &dfResult, NULL);
    }
}

//*** DRBG_InstantiateSeeded()
// This function is used to instantiate a random number generator from seed values.
// The nominal use of this generator is to create sequences of pseudo-random
// numbers from a seed value.
//
// Return Type: TPM_RC
//  TPM_RC_FAILURE      DRBG self-test failure
LIB_EXPORT TPM_RC DRBG_InstantiateSeeded(
    DRBG_STATE*  drbgState,  // IN/OUT: buffer to hold the state
    const TPM2B* seed,       // IN: the seed to use
    const TPM2B* purpose,    // IN: a label for the generation process.
    const TPM2B* name,       // IN: name of the object
    const TPM2B* additional  // IN: additional data
)
{
    DF_STATE dfState;
    int      totalInputSize;
    // DRBG should have been tested, but...
    if(!IsDrbgTested() && !DRBG_SelfTest())
```

```
    {
        FAIL_RC(FATAL_ERROR_SELF_TEST);
    }
    // Initialize the DRBG state
    memset(drbgState, 0, sizeof(DRBG_STATE));
    drbgState->magic = DRBG_MAGIC;

    // Size all of the values
    totalInputSize = (seed != NULL) ? seed->size : 0;
    totalInputSize += (purpose != NULL) ? purpose->size : 0;
    totalInputSize += (name != NULL) ? name->size : 0;
    totalInputSize += (additional != NULL) ? additional->size : 0;

    // Initialize the derivation
    DfStart(&dfState, totalInputSize);

    // Run all the input strings through the derivation function
    if(seed != NULL)
        DfUpdate(&dfState, seed->size, seed->buffer);
    if(purpose != NULL)
        DfUpdate(&dfState, purpose->size, purpose->buffer);
    if(name != NULL)
        DfUpdate(&dfState, name->size, name->buffer);
    if(additional != NULL)
        DfUpdate(&dfState, additional->size, additional->buffer);

    // Used the derivation function output as the "entropy" input. This is not
    // how it is described in SP800-90A but this is the equivalent function
    DRBG_Reseed(((DRBG_STATE*)drbgState), DfEnd(&dfState), NULL);

    return TPM_RC_SUCCESS;
}

//*** CryptRandStartup()
// This function is called when TPM_Startup is executed. This function always returns
// TRUE.
LIB_EXPORT BOOL CryptRandStartup(void)
{
#if !_DRBG_STATE_SAVE
    // If not saved in NV, re-instantiate on each startup
    return DRBG_Instantiate(&drbgDefault, 0, NULL);
#else
    // If the running state is saved in NV, NV has to be loaded before it can
    // be updated
    if(go.drbgState.magic == DRBG_MAGIC)
        return DRBG_Reseed(&go.drbgState, NULL, NULL);
    else
        return DRBG_Instantiate(&go.drbgState, 0, NULL);
#endif
}

//*** CryptRandInit()
// This function is called when _TPM_Init is being processed.
//
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
LIB_EXPORT BOOL CryptRandInit(void)
{
#if !USE_DEBUG_RNG
    _plat__GetEntropy(NULL, 0);
#endif
    return DRBG_SelfTest();
}

//*** DRBG_Generate()
```

```
// This function generates a random sequence according SP800-90A.
// If 'random' is not NULL, then 'randomSize' bytes of random values are generated.
// If 'random' is NULL or 'randomSize' is zero, then the function returns
// zero without generating any bits or updating the reseed counter.
// This function returns the number of bytes produced which could be less than the
// number requested if the request is too large ("too large" is implementation
// dependent.)
LIB_EXPORT UINT16 DRBG_Generate(
    RAND_STATE* state,
    BYTE*       random,     // OUT: buffer to receive the random values
    UINT16      randomSize  // IN: the number of bytes to generate
)
{
    if(state == NULL)
        state = (RAND_STATE*)&drbgDefault;
    if(random == NULL)
        return 0;

    // If the caller used a KDF state, generate a sequence from the KDF not to
    // exceed the limit.
    if(state->kdf.magic == KDF_MAGIC)
    {
        KDF_STATE* kdf       = (KDF_STATE*)state;
        UINT32     counter   = (UINT32)kdf->counter;
        INT32      bytesLeft = randomSize;
        //
        // If the number of bytes to be returned would put the generator
        // over the limit, then return 0
        if((((kdf->counter * kdf->digestSize) + randomSize) * 8) > kdf->limit)
            return 0;
        // Process partial and full blocks until all requested bytes provided
        while(bytesLeft > 0)
        {
            // If there is any residual data in the buffer, copy it to the output
            // buffer
            if(kdf->residual.t.size > 0)
            {
                INT32 size;
                //
                // Don't use more of the residual than will fit or more than are
                // available
                size = MIN(kdf->residual.t.size, bytesLeft);

                // Copy some or all of the residual to the output. The residual is
                // at the end of the buffer. The residual might be a full buffer.
                MemoryCopy(
                    random,
                    &kdf->residual.t.buffer[kdf->digestSize - kdf->residual.t.size],
                    size);

                // Advance the buffer pointer
                random += size;

                // Reduce the number of bytes left to get
                bytesLeft -= size;

                // And reduce the residual size appropriately
                kdf->residual.t.size -= (UINT16)size;
            }
            else
            {
                UINT16 blocks = (UINT16)(bytesLeft / kdf->digestSize);
                //
                // Get the number of required full blocks
                if(blocks > 0)
                {
```

```
                    UINT16 size = blocks * kdf->digestSize;
                    // Get some number of full blocks and put them in the return
buffer
                    CryptKDFa(kdf->hash,
                              kdf->seed,
                              kdf->label,
                              kdf->context,
                              NULL,
                              kdf->limit,
                              random,
                              &counter,
                              blocks);

                    // reduce the size remaining to be moved and advance the pointer
                    bytesLeft -= size;
                    random += size;
                }
                else
                {
                    // Fill the residual buffer with a full block and then loop to
                    // top to get part of it copied to the output.
                    kdf->residual.t.size = CryptKDFa(kdf->hash,
                                                     kdf->seed,
                                                     kdf->label,
                                                     kdf->context,
                                                     NULL,
                                                     kdf->limit,
                                                     kdf->residual.t.buffer,
                                                     &counter,
                                                     1);
                }
            }
        }
        kdf->counter = counter;
        return randomSize;
    }
    else if(state->drbg.magic == DRBG_MAGIC)
    {
        DRBG_STATE*       drbgState = (DRBG_STATE*)state;
        DRBG_KEY_SCHEDULE keySchedule;
        DRBG_SEED*        seed = &drbgState->seed;

        if(drbgState->reseedCounter >= CTR_DRBG_MAX_REQUESTS_PER_RESEED)
        {
            if(drbgState == &drbgDefault)
            {
                DRBG_Reseed(drbgState, NULL, NULL);
                if(IsEntropyBad() && !IsSelfTest())
                    return 0;
            }
            else
            {
                // If this is a PRNG then the only way to get
                // here is if the SW has run away.
                FAIL_IMMEDIATE(FATAL_ERROR_INTERNAL, 0);
            }
        }
        // if the allowed number of bytes in a request is larger than the
        // less than the number of bytes that can be requested, then check
#if UINT16_MAX >= CTR_DRBG_MAX_BYTES_PER_REQUEST
        if(randomSize > CTR_DRBG_MAX_BYTES_PER_REQUEST)
            randomSize = CTR_DRBG_MAX_BYTES_PER_REQUEST;
#endif
        // Create  encryption schedule
        if(DRBG_ENCRYPT_SETUP(
               (BYTE*)pDRBG_KEY(seed), DRBG_KEY_SIZE_BITS, &keySchedule)
```

```
            != 0)
        {
            FAIL_IMMEDIATE(FATAL_ERROR_INTERNAL, 0);
        }
        // Generate the random data
        EncryptDRBG(
            random, randomSize, &keySchedule, pDRBG_IV(seed), drbgState->lastValue);
        // Do a key update
        DRBG_Update(drbgState, &keySchedule, NULL);

        // Increment the reseed counter
        drbgState->reseedCounter += 1;
    }
    else
    {
        // invalid DRBG state structure
        FAIL_IMMEDIATE(FATAL_ERROR_INTERNAL, 0);
    }
    return randomSize;
}

//*** DRBG_Instantiate()
// This is CTR_DRBG_Instantiate_algorithm() from [SP 800-90A 10.2.1.3.1].
// This is called when a the TPM DRBG is to be instantiated. This is
// called to instantiate a DRBG used by the TPM for normal
// operations.
//
//  Return Type: BOOL
//      TRUE(1)         instantiation succeeded
//      FALSE(0)        instantiation failed
LIB_EXPORT BOOL DRBG_Instantiate(
    DRBG_STATE* drbgState,       // OUT: the instantiated value
    UINT16      pSize,           // IN: Size of personalization string
    BYTE*       personalization  // IN: The personalization string
)
{
    DRBG_SEED seed;
    DRBG_SEED dfResult;
    //
    pAssert((pSize == 0) || (pSize <= sizeof(seed)) || (personalization != NULL));
    // If the DRBG has not been tested, test when doing an instantiation. Since
    // Instantiation is called during self test, make sure we don't get stuck in a
    // loop.
    if(!IsDrbgTested() && !IsSelfTest() && !DRBG_SelfTest())
        return FALSE;
    // If doing a self test, DRBG_GetEntropy will return the NIST
    // test vector value.
    if(!DRBG_GetEntropy(sizeof(seed), (BYTE*)&seed))
        return FALSE;
    // set everything to zero
    memset(drbgState, 0, sizeof(DRBG_STATE));
    drbgState->magic = DRBG_MAGIC;

    // Steps 1, 2, 3, 6, 7 of SP 800-90A 10.2.1.3.1 are exactly what
    // reseeding does. So, do a reduction on the personalization value (if any)
    // and do a reseed.
    DRBG_Reseed(drbgState, &seed, DfBuffer(&dfResult, pSize, personalization));

    return TRUE;
}

//*** DRBG_Uninstantiate()
// This is Uninstantiate_function() from [SP 800-90A 9.4].
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE        not a valid state
```

```
LIB_EXPORT TPM_RC DRBG_Uninstantiate(
    DRBG_STATE* drbgState  // IN/OUT: working state to erase
)
{
    if((drbgState == NULL) || (drbgState->magic != DRBG_MAGIC))
        return TPM_RC_VALUE;
    memset(drbgState, 0, sizeof(DRBG_STATE));
    return TPM_RC_SUCCESS;
}
```

7.147 /tpm/src/crypt/CryptRsa.c

```
//** Introduction
//
// This file contains implementation of cryptographic primitives for RSA.
// Vendors may replace the implementation in this file with their own library
// functions.

//**  Includes
// Need this define to get the 'private' defines for this function
#define CRYPT_RSA_C
#include "Tpm.h"
#include "TpmMath_Util_fp.h"

#if ALG_RSA

//**  Obligatory Initialization Functions

//*** CryptRsaInit()
// Function called at _TPM_Init().
BOOL CryptRsaInit(void)
{
    return TRUE;
}

//*** CryptRsaStartup()
// Function called at TPM2_Startup()
BOOL CryptRsaStartup(void)
{
    return TRUE;
}

//** Internal Functions

//*** RsaInitializeExponent()
// This function initializes the bignum data structure that holds the private
// exponent. This function returns the pointer to the private exponent value so that
// it can be used in an initializer for a data declaration.

static privateExponent* RsaInitializeExponent(privateExponent* Z)
{
    // verify privateExponent packing matches the usage of the bn pointer as an
    // array in below function
    MUST_BE(offsetof(privateExponent, Q) == SIZEOF_MEMBER(privateExponent, P));

    Crypt_Int** bn = (Crypt_Int**)&Z->P;
    int         i;
    //
    for(i = 0; i < 5; i++)
    {
        bn[i] = (Crypt_Int*)&(Z->entries[i]);
        ExtMath_Initialize_Int(bn[i], MAX_RSA_KEY_BITS / 2);
    }
    return Z;
}
```

```
//*** MakePgreaterThanQ()
// This function swaps the pointers for P and Q if Q happens to be larger than Q.
static void MakePgreaterThanQ(privateExponent* Z)
{
    if(ExtMath_UnsignedCmp(Z->P, Z->Q) < 0)
    {
        Crypt_Int* bnT = Z->P;
        Z->P           = Z->Q;
        Z->Q           = bnT;
    }
}

//*** PackExponent()
// This function takes the bignum private exponent and converts it into TPM2B form.
// In this form, the size field contains the overall size of the packed data. The
// buffer contains 5, equal sized values in P, Q, dP, dQ, qInv order. For example, if
// a key has a 2Kb public key, then the packed private key will contain 5, 1Kb values.
// This form makes it relatively easy to load and save the values without changing
// the normal unmarshaling to do anything more than allow a larger TPM2B for the
// private key. Also, when exporting the value, all that is needed is to change the
// size field of the private key in order to save just the P value.
//  Return Type: BOOL
//      TRUE(1)     success
//      FALSE(0)    failure         // The data is too big to fit
static BOOL PackExponent(TPM2B_PRIVATE_KEY_RSA* packed, privateExponent* Z)
{
    int    i;
    UINT16 primeSize = (UINT16)BITS_TO_BYTES(ExtMath_MostSigBitNum(Z->P));
    UINT16 pS        = primeSize;
    //
    pAssert((primeSize * 5) <= sizeof(packed->t.buffer));
    packed->t.size = (primeSize * 5) + RSA_prime_flag;
    for(i = 0; i < 5; i++)
        if(!ExtMath_IntToBytes(
                (Crypt_Int*)&Z->entries[i], &packed->t.buffer[primeSize * i], &pS))
            return FALSE;
    if(pS != primeSize)
        return FALSE;
    return TRUE;
}

//*** UnpackExponent()
// This function unpacks the private exponent from its TPM2B form into its bignum
// form.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        TPM2B is not the correct size
static BOOL UnpackExponent(TPM2B_PRIVATE_KEY_RSA* b, privateExponent* Z)
{
    UINT16      primeSize = b->t.size & ~RSA_prime_flag;
    int         i;
    Crypt_Int** bn = &Z->P;
    //
    GOTO_ERROR_UNLESS(b->t.size & RSA_prime_flag);
    RsaInitializeExponent(Z);
    GOTO_ERROR_UNLESS((primeSize % 5) == 0);
    primeSize /= 5;
    for(i = 0; i < 5; i++)
        GOTO_ERROR_UNLESS(
            ExtMath_IntFromBytes(bn[i], &b->t.buffer[primeSize * i], primeSize)
            != NULL);
    MakePgreaterThanQ(Z);
    return TRUE;
Error:
    return FALSE;
```

```
}

//*** ComputePrivateExponent()
// This function computes the private exponent from the primes.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
static BOOL ComputePrivateExponent(
    Crypt_Int*       pubExp,  // IN: the public exponent
    privateExponent* Z        // IN/OUT: on input, has primes P and Q. On
                              //         output, has P, Q, dP, dQ, and pInv
)
{
    BOOL pOK;
    BOOL qOK;
    CRYPT_PRIME_VAR(pT);
    //
    // make p the larger value so that m2 is always less than p
    MakePgreaterThanQ(Z);

    //dP = (1/e) mod (p-1)
    pOK = ExtMath_SubtractWord(pT, Z->P, 1);
    pOK = pOK && ExtMath_ModInverse(Z->dP, pubExp, pT);
    //dQ = (1/e) mod (q-1)
    qOK = ExtMath_SubtractWord(pT, Z->Q, 1);
    qOK = qOK && ExtMath_ModInverse(Z->dQ, pubExp, pT);
    // qInv = (1/q) mod p
    if(pOK && qOK)
        pOK = qOK = ExtMath_ModInverse(Z->qInv, Z->Q, Z->P);
    if(!pOK)
        ExtMath_SetWord(Z->P, 0);
    if(!qOK)
        ExtMath_SetWord(Z->Q, 0);
    return pOK && qOK;
}

//*** RsaPrivateKeyOp()
// This function is called to do the exponentiation with the private key. Compile
// options allow use of the simple (but slow) private exponent, or the more complex
// but faster CRT method.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
static BOOL RsaPrivateKeyOp(Crypt_Int* inOut,  // IN/OUT: number to be exponentiated
                            privateExponent* Z)
{
    CRYPT_RSA_VAR(M1);
    CRYPT_RSA_VAR(M2);
    CRYPT_RSA_VAR(M);
    CRYPT_RSA_VAR(H);
    //
    MakePgreaterThanQ(Z);
    // m1 = cdP mod p
    GOTO_ERROR_UNLESS(ExtMath_ModExp(M1, inOut, Z->dP, Z->P));
    // m2 = cdQ mod q
    GOTO_ERROR_UNLESS(ExtMath_ModExp(M2, inOut, Z->dQ, Z->Q));
    // h = qInv * (m1 - m2) mod p = qInv * (m1 + P - m2) mod P because Q < P
    // so m2 < P
    GOTO_ERROR_UNLESS(ExtMath_Subtract(H, Z->P, M2));
    GOTO_ERROR_UNLESS(ExtMath_Add(H, H, M1));
    GOTO_ERROR_UNLESS(ExtMath_ModMult(H, H, Z->qInv, Z->P));
    // m = m2 + h * q
    GOTO_ERROR_UNLESS(ExtMath_Multiply(M, H, Z->Q));
    GOTO_ERROR_UNLESS(ExtMath_Add(inOut, M2, M));
    return TRUE;
Error:
```

```
    return FALSE;
}

//*** RSAEP()
// This function performs the RSAEP operation defined in PKCS#1v2.1. It is
// an exponentiation of a value ('m') with the public exponent ('e'), modulo
// the public ('n').
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE     number to exponentiate is larger than the modulus
//
static TPM_RC RSAEP(TPM2B* dInOut,  // IN: size of the encrypted block and the size of
                                    //     the encrypted value. It must be the size of
                                    //     the modulus.
                                    // OUT: the encrypted data. Will receive the
                                    //      decrypted value
                    OBJECT* key     // IN: the key to use
)
{
    TPM2B_TYPE(4BYTES, 4);
    TPM2B_4BYTES e2B;
    UINT32       e = key->publicArea.parameters.rsaDetail.exponent;
    //
    if(e == 0)
        e = RSA_DEFAULT_PUBLIC_EXPONENT;
    UINT32_TO_BYTE_ARRAY(e, e2B.t.buffer);
    e2B.t.size = 4;
    return ModExpB(dInOut->size,
                   dInOut->buffer,
                   dInOut->size,
                   dInOut->buffer,
                   e2B.t.size,
                   e2B.t.buffer,
                   key->publicArea.unique.rsa.t.size,
                   key->publicArea.unique.rsa.t.buffer);
}

//*** RSADP()
// This function performs the RSADP operation defined in PKCS#1v2.1. It is
// an exponentiation of a value ('c') with the private exponent ('d'), modulo
// the public modulus ('n'). The decryption is in place.
//
// This function also checks the size of the private key. If the size indicates
// that only a prime value is present, the key is converted to being a private
// exponent.
//
//  Return Type: TPM_RC
//      TPM_RC_SIZE         the value to decrypt is larger than the modulus
//
static TPM_RC RSADP(TPM2B*  inOut,  // IN/OUT: the value to encrypt
                    OBJECT* key     // IN: the key
)
{
    CRYPT_RSA_INITIALIZED(bnM, inOut);
    NEW_PRIVATE_EXPONENT(Z);
    if(UnsignedCompareB(inOut->size,
                        inOut->buffer,
                        key->publicArea.unique.rsa.t.size,
                        key->publicArea.unique.rsa.t.buffer)
       >= 0)
        return TPM_RC_SIZE;
    // private key operation requires that private exponent be loaded
    // During self-test, this might not be the case so load it up if it hasn't
    // already done
    // been done
    if((key->sensitive.sensitive.rsa.t.size & RSA_prime_flag) == 0)
```

```
    {
        if(CryptRsaLoadPrivateExponent(&key->publicArea, &key->sensitive)
           != TPM_RC_SUCCESS)
            return TPM_RC_BINDING;
    }
    GOTO_ERROR_UNLESS(UnpackExponent(&key->sensitive.sensitive.rsa, Z));
    GOTO_ERROR_UNLESS(RsaPrivateKeyOp(bnM, Z));
    GOTO_ERROR_UNLESS(TpmMath_IntTo2B(bnM, inOut, inOut->size));
    return TPM_RC_SUCCESS;
Error:
    return TPM_RC_FAILURE;
}

//*** OaepEncode()
// This function performs OAEP padding. The size of the buffer to receive the
// OAEP padded data must equal the size of the modulus
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE     'hashAlg' is not valid or message size is too large
//
static TPM_RC OaepEncode(
    TPM2B*       padded,   // OUT: the pad data
    TPM_ALG_ID   hashAlg,  // IN: algorithm to use for padding
    const TPM2B* label,    // IN: null-terminated string (may be NULL)
    TPM2B*       message,  // IN: the message being padded
    RAND_STATE*  rand      // IN: the random number generator to use
)
{
    INT32  padLen;
    INT32  dbSize;
    INT32  i;
    BYTE   mySeed[MAX_DIGEST_SIZE];
    BYTE*  seed = mySeed;
    UINT16 hLen = CryptHashGetDigestSize(hashAlg);
    BYTE   mask[MAX_RSA_KEY_BYTES];
    BYTE*  pp;
    BYTE*  pm;
    TPM_RC retVal = TPM_RC_SUCCESS;

    pAssert(padded != NULL && message != NULL);

    // A value of zero is not allowed because the KDF can't produce a result
    // if the digest size is zero.
    if(hLen == 0)
        return TPM_RC_VALUE;

    // Basic size checks
    //  make sure digest isn't too big for key size
    if(padded->size < (2 * hLen) + 2)
        ERROR_EXIT(TPM_RC_HASH);

    // and that message will fit messageSize <= k - 2hLen - 2
    if(message->size > (padded->size - (2 * hLen) - 2))
        ERROR_EXIT(TPM_RC_VALUE);

    // Hash L even if it is null
    // Offset into padded leaving room for masked seed and byte of zero
    pp = &padded->buffer[hLen + 1];
    if(CryptHashBlock(hashAlg, label->size, (BYTE*)label->buffer, hLen, pp) != hLen)
        ERROR_EXIT(TPM_RC_FAILURE);

    // concatenate PS of k  mLen  2hLen  2
    padLen = padded->size - message->size - (2 * hLen) - 2;
    MemorySet(&pp[hLen], 0, padLen);
    pp[hLen + padLen] = 0x01;
    padLen += 1;
```

```
    memcpy(&pp[hLen + padLen], message->buffer, message->size);

    // The total size of db = hLen + pad + mSize;
    dbSize = hLen + padLen + message->size;

    DRBG_Generate(rand, mySeed, (UINT16)hLen);
    if(g_inFailureMode)
        ERROR_EXIT(TPM_RC_FAILURE);
    // mask = MGF1 (seed, nSize  hLen  1)
    CryptMGF_KDF(dbSize, mask, hashAlg, hLen, seed, 0);

    // Create the masked db
    pm = mask;
    for(i = dbSize; i > 0; i--)
        *pp++ ^= *pm++;
    pp = &padded->buffer[hLen + 1];

    // Run the masked data through MGF1
    if(CryptMGF_KDF(hLen, &padded->buffer[1], hashAlg, dbSize, pp, 0)
       != (unsigned)hLen)
        ERROR_EXIT(TPM_RC_VALUE);
    // Now XOR the seed to create masked seed
    pp = &padded->buffer[1];
    pm = seed;
    for(i = hLen; i > 0; i--)
        *pp++ ^= *pm++;
    // Set the first byte to zero
    padded->buffer[0] = 0x00;
Exit:
    return retVal;
}

//*** OaepDecode()
// This function performs OAEP padding checking. The size of the buffer to receive
// the recovered data. If the padding is not valid, the 'dSize' size is set to zero
// and the function returns TPM_RC_VALUE.
//
// The 'dSize' parameter is used as an input to indicate the size available in the
// buffer.

// If insufficient space is available, the size is not changed and the return code
// is TPM_RC_VALUE.
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE        the value to decode was larger than the modulus, or
//                          the padding is wrong or the buffer to receive the
//                          results is too small
//
//
static TPM_RC OaepDecode(
    TPM2B*       dataOut,  // OUT: the recovered data
    TPM_ALG_ID   hashAlg,  // IN: algorithm to use for padding
    const TPM2B* label,    // IN: null-terminated string (may be NULL)
    TPM2B*       padded    // IN: the padded data
)
{
    UINT32 i;
    BYTE   seedMask[MAX_DIGEST_SIZE];
    UINT32 hLen = CryptHashGetDigestSize(hashAlg);

    BYTE   mask[MAX_RSA_KEY_BYTES];
    BYTE*  pp;
    BYTE*  pm;
    TPM_RC retVal = TPM_RC_SUCCESS;

    // Strange size (anything smaller can't be an OAEP padded block)
```

```
    // Also check for no leading 0
    if((padded->size < (unsigned)((2 * hLen) + 2)) || (padded->buffer[0] != 0))
        ERROR_EXIT(TPM_RC_VALUE);
    // Use the hash size to determine what to put through MGF1 in order
    // to recover the seedMask
    CryptMGF_KDF(hLen,
                 seedMask,
                 hashAlg,
                 padded->size - hLen - 1,
                 &padded->buffer[hLen + 1],
                 0);

    // Recover the seed into seedMask
    pAssert(hLen <= sizeof(seedMask));
    pp = &padded->buffer[1];
    pm = seedMask;
    for(i = hLen; i > 0; i--)
        *pm++ ^= *pp++;

    // Use the seed to generate the data mask
    CryptMGF_KDF(padded->size - hLen - 1, mask, hashAlg, hLen, seedMask, 0);

    // Use the mask generated from seed to recover the padded data
    pp = &padded->buffer[hLen + 1];
    pm = mask;
    for(i = (padded->size - hLen - 1); i > 0; i--)
        *pm++ ^= *pp++;

    // Make sure that the recovered data has the hash of the label
    // Put trial value in the seed mask
    if((CryptHashBlock(hashAlg, label->size, (BYTE*)label->buffer, hLen, seedMask))
       != hLen)
        FAIL(FATAL_ERROR_INTERNAL);
    if(memcmp(seedMask, mask, hLen) != 0)
        ERROR_EXIT(TPM_RC_VALUE);

    // find the start of the data
    pm = &mask[hLen];
    for(i = (UINT32)padded->size - (2 * hLen) - 1; i > 0; i--)
    {
        if(*pm++ != 0)
            break;
    }
    // If we ran out of data or didn't end with 0x01, then return an error
    if(i == 0 || pm[-1] != 0x01)
        ERROR_EXIT(TPM_RC_VALUE);

    // pm should be pointing at the first part of the data
    // and i is one greater than the number of bytes to move
    i--;
    if(i > dataOut->size)
        // Special exit to preserve the size of the output buffer
        return TPM_RC_VALUE;
    memcpy(dataOut->buffer, pm, i);
    dataOut->size = (UINT16)i;
Exit:
    if(retVal != TPM_RC_SUCCESS)
        dataOut->size = 0;
    return retVal;
}

//*** PKCS1v1_5Encode()
// This function performs the encoding for RSAES-PKCS1-V1_5-ENCRYPT as defined in
// PKCS#1V2.1
//  Return Type: TPM_RC
//      TPM_RC_VALUE     message size is too large
```

```
//
static TPM_RC RSAES_PKCS1v1_5Encode(TPM2B* padded,   // OUT: the pad data
                                    TPM2B* message,  // IN: the message being padded
                                    RAND_STATE* rand)
{
    UINT32 ps = padded->size - message->size - 3;
    //
    if(message->size > padded->size - 11)
        return TPM_RC_VALUE;
    // move the message to the end of the buffer
    memcpy(&padded->buffer[padded->size - message->size],
           message->buffer,
           message->size);
    // Set the first byte to 0x00 and the second to 0x02
    padded->buffer[0] = 0;
    padded->buffer[1] = 2;

    // Fill with random bytes
    DRBG_Generate(rand, &padded->buffer[2], (UINT16)ps);
    if(g_inFailureMode)
        return TPM_RC_FAILURE;

    // Set the delimiter for the random field to 0
    padded->buffer[2 + ps] = 0;

    // Now, the only messy part. Make sure that all the 'ps' bytes are non-zero
    // In this implementation, use the value of the current index
    for(ps++; ps > 1; ps--)
    {
        if(padded->buffer[ps] == 0)
            padded->buffer[ps] = 0x55;  // In the < 0.5% of the cases that the
                                        // random value is 0, just pick a value to
                                        // put into the spot.
    }
    return TPM_RC_SUCCESS;
}

//*** RSAES_Decode()
// This function performs the decoding for RSAES-PKCS1-V1_5-ENCRYPT as defined in
// PKCS#1V2.1
//
//  Return Type: TPM_RC
//      TPM_RC_FAIL      decoding error or results would no fit into provided buffer
//
static TPM_RC RSAES_Decode(TPM2B* message,  // OUT: the recovered message
                           TPM2B* coded     // IN: the encoded message
)
{
    BOOL   fail = FALSE;
    UINT16 pSize;

    fail = (coded->size < 11);
    fail = (coded->buffer[0] != 0x00) | fail;
    fail = (coded->buffer[1] != 0x02) | fail;
    for(pSize = 2; pSize < coded->size; pSize++)
    {
        if(coded->buffer[pSize] == 0)
            break;
    }
    pSize++;

    // Make sure that pSize has not gone over the end and that there are at least 8
    // bytes of pad data.
    fail = (pSize > coded->size) | fail;
    fail = ((pSize - 2) <= 8) | fail;
    if((message->size < (UINT16)(coded->size - pSize)) || fail)
```

```
        return TPM_RC_VALUE;
    message->size = coded->size - pSize;
    memcpy(message->buffer, &coded->buffer[pSize], coded->size - pSize);
    return TPM_RC_SUCCESS;
}

//*** CryptRsaPssSaltSize()
// This function computes the salt size used in PSS. It is broken out so that
// the X509 code can get the same value that is used by the encoding function in this
// module.
INT16
CryptRsaPssSaltSize(INT16 hashSize, INT16 outSize)
{
    INT16 saltSize;
    //
    // (Mask Length) = (outSize - hashSize - 1);
    // Max saltSize is (Mask Length) - 1
    saltSize = (outSize - hashSize - 1) - 1;
    // Use the maximum salt size allowed by FIPS 186-4
    if(saltSize > hashSize)
        saltSize = hashSize;
    else if(saltSize < 0)
        saltSize = 0;
    return saltSize;
}

//*** PssEncode()
// This function creates an encoded block of data that is the size of modulus.
// The function uses the maximum salt size that will fit in the encoded block.
//
//  Returns TPM_RC_SUCCESS or goes into failure mode.
static TPM_RC PssEncode(TPM2B*      out,      // OUT: the encoded buffer
                        TPM_ALG_ID  hashAlg,  // IN: hash algorithm for the encoding
                        TPM2B*      digest,   // IN: the digest
                        RAND_STATE* rand      // IN: random number source
)
{
    UINT32     hLen = CryptHashGetDigestSize(hashAlg);
    BYTE       salt[MAX_RSA_KEY_BYTES - 1];
    UINT16     saltSize;
    BYTE*      ps = salt;
    BYTE*      pOut;
    UINT16     mLen;
    HASH_STATE hashState;

    // These are fatal errors indicating bad TPM firmware
    pAssert(out != NULL && hLen > 0 && digest != NULL);

    // Get the size of the mask
    mLen = (UINT16)(out->size - hLen - 1);

    // Set the salt size
    saltSize = CryptRsaPssSaltSize((INT16)hLen, (INT16)out->size);

    //using eOut for scratch space
    // Set the first 8 bytes to zero
    pOut = out->buffer;
    memset(pOut, 0, 8);

    // Get set the salt
    DRBG_Generate(rand, salt, saltSize);
    if(g_inFailureMode)
        return TPM_RC_FAILURE;

    // Create the hash of the pad || input hash || salt
    CryptHashStart(&hashState, hashAlg);
```

```
    CryptDigestUpdate(&hashState, 8, pOut);
    CryptDigestUpdate2B(&hashState, digest);
    CryptDigestUpdate(&hashState, saltSize, salt);
    CryptHashEnd(&hashState, hLen, &pOut[out->size - hLen - 1]);

    // Create a mask
    if(CryptMGF_KDF(mLen, pOut, hashAlg, hLen, &pOut[mLen], 0) != mLen)
        FAIL(FATAL_ERROR_INTERNAL);

    // Since this implementation uses key sizes that are all even multiples of
    // 8, just need to make sure that the most significant bit is CLEAR
    *pOut &= 0x7f;

    // Before we mess up the pOut value, set the last byte to 0xbc
    pOut[out->size - 1] = 0xbc;

    // XOR a byte of 0x01 at the position just before where the salt will be XOR'ed
    pOut = &pOut[mLen - saltSize - 1];
    *pOut++ ^= 0x01;

    // XOR the salt data into the buffer
    for(; saltSize > 0; saltSize--)
        *pOut++ ^= *ps++;

    // and we are done
    return TPM_RC_SUCCESS;
}

//*** PssDecode()
// This function checks that the PSS encoded block was built from the
// provided digest. If the check is successful, TPM_RC_SUCCESS is returned.
// Any other value indicates an error.
//
// This implementation of PSS decoding is intended for the reference TPM
// implementation and is not at all generalized.  It is used to check
// signatures over hashes and assumptions are made about the sizes of values.
// Those assumptions are enforce by this implementation.
// This implementation does allow for a variable size salt value to have been
// used by the creator of the signature.
//
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       'hashAlg' is not a supported hash algorithm
//      TPM_RC_VALUE         decode operation failed
//
static TPM_RC PssDecode(
    TPM_ALG_ID hashAlg,  // IN: hash algorithm to use for the encoding
    TPM2B*     dIn,      // In: the digest to compare
    TPM2B*     eIn       // IN: the encoded data
)
{
    UINT32     hLen = CryptHashGetDigestSize(hashAlg);
    BYTE       mask[MAX_RSA_KEY_BYTES];
    BYTE*      pm = mask;
    BYTE*      pe;
    BYTE       pad[8] = {0};
    UINT32     i;
    UINT32     mLen;
    BYTE       fail;
    TPM_RC     retVal = TPM_RC_SUCCESS;
    HASH_STATE hashState;

    // These errors are indicative of failures due to programmer error
    pAssert(dIn != NULL && eIn != NULL);
    pe = eIn->buffer;

    // check the hash scheme
```

```
    if(hLen == 0)
        ERROR_EXIT(TPM_RC_SCHEME);

    // most significant bit must be zero
    fail = pe[0] & 0x80;

    // last byte must be 0xbc
    fail |= pe[eIn->size - 1] ^ 0xbc;

    // Use the hLen bytes at the end of the buffer to generate a mask
    // Doesn't start at the end which is a flag byte
    mLen = eIn->size - hLen - 1;
    CryptMGF_KDF(mLen, mask, hashAlg, hLen, &pe[mLen], 0);

    // Clear the MSO of the mask to make it consistent with the encoding.
    mask[0] &= 0x7F;

    pAssert(mLen <= sizeof(mask));
    // XOR the data into the mask to recover the salt. This sequence
    // advances eIn so that it will end up pointing to the seed data
    // which is the hash of the signature data
    for(i = mLen; i > 0; i--)
        *pm++ ^= *pe++;

    // Find the first byte of 0x01 after a string of all 0x00
    for(pm = mask, i = mLen; i > 0; i--)
    {
        if(*pm == 0x01)
            break;
        else
            fail |= *pm++;
    }
    // i should not be zero
    fail |= (i == 0);

    // if we have failed, will continue using the entire mask as the salt value so
    // that the timing attacks will not disclose anything (I don't think that this
    // is a problem for TPM applications but, usually, we don't fail so this
    // doesn't cost anything).
    if(fail)
    {
        i  = mLen;
        pm = mask;
    }
    else
    {
        pm++;
        i--;
    }
    // i contains the salt size and pm points to the salt. Going to use the input
    // hash and the seed to recreate the hash in the lower portion of eIn.
    CryptHashStart(&hashState, hashAlg);

    // add the pad of 8 zeros
    CryptDigestUpdate(&hashState, 8, pad);

    // add the provided digest value
    CryptDigestUpdate(&hashState, dIn->size, dIn->buffer);

    // and the salt
    CryptDigestUpdate(&hashState, i, pm);

    // get the result
    fail |= (CryptHashEnd(&hashState, hLen, mask) != hLen);

    // Compare all bytes
```

```
    for(pm = mask; hLen > 0; hLen--)
        // don't use fail = because that could skip the increment and compare
        // operations after the first failure and that gives away timing
        // information.
        fail |= *pm++ ^ *pe++;

    retVal = (fail != 0) ? TPM_RC_VALUE : TPM_RC_SUCCESS;
Exit:
    return retVal;
}

//*** MakeDerTag()
// Construct the DER value that is used in RSASSA
//  Return Type: INT16
//   > 0        size of value
//   <= 0       no hash exists
INT16
MakeDerTag(TPM_ALG_ID hashAlg, INT16 sizeOfBuffer, BYTE* buffer)
{
    //    0x30, 0x31,       // SEQUENCE (2 elements) 1st
    //        0x30, 0x0D,   // SEQUENCE (2 elements)
    //            0x06, 0x09,   // HASH OID
    //                0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x01,
    //             0x05, 0x00,  // NULL
    //        0x04, 0x20  //  OCTET STRING
    HASH_DEF* info = CryptGetHashDef(hashAlg);
    INT16     oidSize;
    // If no OID, can't do encode
    GOTO_ERROR_UNLESS(info != NULL);
    oidSize = 2 + (info->OID)[1];
    // make sure this fits in the buffer
    GOTO_ERROR_UNLESS(sizeOfBuffer >= (oidSize + 8));
    *buffer++ = 0x30;  // 1st SEQUENCE
    // Size of the 1st SEQUENCE is 6 bytes + size of the hash OID + size of the
    // digest size
    *buffer++ = (BYTE)(6 + oidSize + info->digestSize);  //
    *buffer++ = 0x30;                                    // 2nd SEQUENCE
    // size is 4 bytes of overhead plus the side of the OID
    *buffer++ = (BYTE)(2 + oidSize);
    MemoryCopy(buffer, info->OID, oidSize);
    buffer += oidSize;
    *buffer++ = 0x05;  // Add a NULL
    *buffer++ = 0x00;

    *buffer++ = 0x04;
    *buffer++ = (BYTE)(info->digestSize);
    return oidSize + 8;
Error:
    return 0;
}

//*** RSASSA_Encode()
// Encode a message using PKCS1v1.5 method.
//
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       'hashAlg' is not a supported hash algorithm
//      TPM_RC_SIZE         'eOutSize' is not large enough
//      TPM_RC_VALUE        'hInSize' does not match the digest size of hashAlg
static TPM_RC RSASSA_Encode(TPM2B* pOut,  // IN:OUT on in, the size of the public key
                                          //        on out, the encoded area
                            TPM_ALG_ID hashAlg,  // IN: hash algorithm for PKCS1v1_5
                            TPM2B*     hIn       // IN: digest value to encode
)
{
    BYTE   DER[20];
    BYTE*  der     = DER;
```

```
    INT32  derSize = MakeDerTag(hashAlg, sizeof(DER), DER);
    BYTE*  eOut;
    INT32  fillSize;
    TPM_RC retVal = TPM_RC_SUCCESS;

    // Can't use this scheme if the algorithm doesn't have a DER string defined.
    if(derSize == 0)
        ERROR_EXIT(TPM_RC_SCHEME);

    // If the digest size of 'hashAl' doesn't match the input digest size, then
    // the DER will misidentify the digest so return an error
    if(CryptHashGetDigestSize(hashAlg) != hIn->size)
        ERROR_EXIT(TPM_RC_VALUE);
    fillSize = pOut->size - derSize - hIn->size - 3;
    eOut     = pOut->buffer;

    // Make sure that this combination will fit in the provided space
    if(fillSize < 8)
        ERROR_EXIT(TPM_RC_SIZE);

    // Start filling
    *eOut++ = 0;  // initial byte of zero
    *eOut++ = 1;  // byte of 0x01
    for(; fillSize > 0; fillSize--)
        *eOut++ = 0xff;  // bunch of 0xff
    *eOut++ = 0;         // another 0
    for(; derSize > 0; derSize--)
        *eOut++ = *der++;  // copy the DER
    der = hIn->buffer;
    for(fillSize = hIn->size; fillSize > 0; fillSize--)
        *eOut++ = *der++;  // copy the hash
Exit:
    return retVal;
}

//*** RSASSA_Decode()
// This function performs the RSASSA decoding of a signature.
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE          decode unsuccessful
//      TPM_RC_SCHEME        'haslAlg' is not supported
//
static TPM_RC RSASSA_Decode(
    TPM_ALG_ID hashAlg,  // IN: hash algorithm to use for the encoding
    TPM2B*     hIn,      // In: the digest to compare
    TPM2B*     eIn       // IN: the encoded data
)
{
    BYTE   fail;
    BYTE   DER[20];
    BYTE*  der     = DER;
    INT32  derSize = MakeDerTag(hashAlg, sizeof(DER), DER);
    BYTE*  pe;
    INT32  hashSize = CryptHashGetDigestSize(hashAlg);
    INT32  fillSize;
    TPM_RC retVal;
    BYTE*  digest;
    UINT16 digestSize;

    pAssert(hIn != NULL && eIn != NULL);
    pe = eIn->buffer;

    // Can't use this scheme if the algorithm doesn't have a DER string
    // defined or if the provided hash isn't the right size
    if(derSize == 0 || (unsigned)hashSize != hIn->size)
        ERROR_EXIT(TPM_RC_SCHEME);
```

```
    // Make sure that this combination will fit in the provided space
    // Since no data movement takes place, can just walk though this
    // and accept nearly random values. This can only be called from
    // CryptValidateSignature() so eInSize is known to be in range.
    fillSize = eIn->size - derSize - hashSize - 3;

    // Start checking (fail will become non-zero if any of the bytes do not have
    // the expected value.
    fail = *pe++;       // initial byte of zero
    fail |= *pe++ ^ 1;  // byte of 0x01
    for(; fillSize > 0; fillSize--)
        fail |= *pe++ ^ 0xff;  // bunch of 0xff
    fail |= *pe++;             // another 0
    for(; derSize > 0; derSize--)
        fail |= *pe++ ^ *der++;  // match the DER
    digestSize = hIn->size;
    digest     = hIn->buffer;
    for(; digestSize > 0; digestSize--)
        fail |= *pe++ ^ *digest++;  // match the hash
    retVal = (fail != 0) ? TPM_RC_VALUE : TPM_RC_SUCCESS;
Exit:
    return retVal;
}

//** Externally Accessible Functions

//*** CryptRsaSelectScheme()
// This function is used by TPM2_RSA_Decrypt and TPM2_RSA_Encrypt.  It sets up
// the rules to select a scheme between input and object default.
// This function assume the RSA object is loaded.
// If a default scheme is defined in object, the default scheme should be chosen,
// otherwise, the input scheme should be chosen.
// In the case that both the object and 'scheme' are not TPM_ALG_NULL, then
// if the schemes are the same, the input scheme will be chosen.
// if the scheme are not compatible, a NULL pointer will be returned.
//
// The return pointer may point to a TPM_ALG_NULL scheme.
TPMT_RSA_DECRYPT* CryptRsaSelectScheme(
    TPMI_DH_OBJECT    rsaHandle,  // IN: handle of an RSA key
    TPMT_RSA_DECRYPT* scheme      // IN: a sign or decrypt scheme
)
{
    OBJECT*           rsaObject;
    TPMT_ASYM_SCHEME* keyScheme;
    TPMT_RSA_DECRYPT* retVal = NULL;

    // Get sign object pointer
    rsaObject = HandleToObject(rsaHandle);
    keyScheme = &rsaObject->publicArea.parameters.asymDetail.scheme;

    // if the default scheme of the object is TPM_ALG_NULL, then select the
    // input scheme
    if(keyScheme->scheme == TPM_ALG_NULL)
    {
        retVal = scheme;
    }
    // if the object scheme is not TPM_ALG_NULL and the input scheme is
    // TPM_ALG_NULL, then select the default scheme of the object.
    else if(scheme->scheme == TPM_ALG_NULL)
    {
        // if input scheme is NULL
        retVal = (TPMT_RSA_DECRYPT*)keyScheme;
    }
    // get here if both the object scheme and the input scheme are
    // not TPM_ALG_NULL. Need to insure that they are the same.
```

```
    // IMPLEMENTATION NOTE: This could cause problems if future versions have
    // schemes that have more values than just a hash algorithm. A new function
    // (IsSchemeSame()) might be needed then.
    else if(keyScheme->scheme == scheme->scheme
            && keyScheme->details.anySig.hashAlg == scheme->details.anySig.hashAlg)
    {
        retVal = scheme;
    }
    // two different, incompatible schemes specified will return NULL
    return retVal;
}

//*** CryptRsaLoadPrivateExponent()
// This function is called to generate the private exponent of an RSA key.
//  Return Type: TPM_RC
//      TPM_RC_BINDING      public and private parts of 'rsaKey' are not matched
TPM_RC
CryptRsaLoadPrivateExponent(TPMT_PUBLIC* publicArea, TPMT_SENSITIVE* sensitive)
{
    //
    if((sensitive->sensitive.rsa.t.size & RSA_prime_flag) == 0)
    {
        if((sensitive->sensitive.rsa.t.size * 2) == publicArea->unique.rsa.t.size)
        {
            NEW_PRIVATE_EXPONENT(Z);
            CRYPT_RSA_INITIALIZED(bnN, &publicArea->unique.rsa);
            CRYPT_RSA_VAR(bnQr);
            CRYPT_INT_VAR(bnE, RADIX_BITS);

            TPM_DO_SELF_TEST(TPM_ALG_NULL);

            GOTO_ERROR_UNLESS((sensitive->sensitive.rsa.t.size * 2)
                              == publicArea->unique.rsa.t.size);
            // Initialize the exponent
            ExtMath_SetWord(bnE, publicArea->parameters.rsaDetail.exponent);
            if(ExtMath_IsZero(bnE))
                ExtMath_SetWord(bnE, RSA_DEFAULT_PUBLIC_EXPONENT);
            // Convert first prime to 2B
            GOTO_ERROR_UNLESS(
                TpmMath_IntFrom2B(Z->P, &sensitive->sensitive.rsa.b) != NULL);

            // Find the second prime by division. This uses 'bQ' rather than Z->Q
            // because the division could make the quotient larger than a prime during
            // some intermediate step.
            GOTO_ERROR_UNLESS(ExtMath_Divide(Z->Q, bnQr, bnN, Z->P));
            GOTO_ERROR_UNLESS(ExtMath_IsZero(bnQr));
            // Compute the private exponent and return it if found
            GOTO_ERROR_UNLESS(ComputePrivateExponent(bnE, Z));
            GOTO_ERROR_UNLESS(PackExponent(&sensitive->sensitive.rsa, Z));
        }
        else
            GOTO_ERROR_UNLESS(((sensitive->sensitive.rsa.t.size / 5) * 2)
                              == publicArea->unique.rsa.t.size);
        sensitive->sensitive.rsa.t.size |= RSA_prime_flag;
    }
    return TPM_RC_SUCCESS;
Error:
    return TPM_RC_BINDING;
}

//*** CryptRsaEncrypt()
// This is the entry point for encryption using RSA. Encryption is
// use of the public exponent. The padding parameter determines what
// padding will be used.
//
// The 'cOutSize' parameter must be at least as large as the size of the key.
```

```
//
// If the padding is RSA_PAD_NONE, 'dIn' is treated as a number. It must be
// lower in value than the key modulus.
// NOTE: If dIn has fewer bytes than cOut, then we don't add low-order zeros to
//       dIn to make it the size of the RSA key for the call to RSAEP. This is
//       because the high order bytes of dIn might have a numeric value that is
//       greater than the value of the key modulus. If this had low-order zeros
//       added, it would have a numeric value larger than the modulus even though
//       it started out with a lower numeric value.
//
//  Return Type: TPM_RC
//      TPM_RC_VALUE     'cOutSize' is too small (must be the size
//                        of the modulus)
//      TPM_RC_SCHEME    'padType' is not a supported scheme
//
LIB_EXPORT TPM_RC CryptRsaEncrypt(
    TPM2B_PUBLIC_KEY_RSA* cOut,     // OUT: the encrypted data
    TPM2B*                dIn,      // IN: the data to encrypt
    OBJECT*               key,      // IN: the key used for encryption
    TPMT_RSA_DECRYPT*     scheme,   // IN: the type of padding and hash
                                    //     if needed
    const TPM2B* label,             // IN: in case it is needed
    RAND_STATE*  rand               // IN: random number generator
                                    //     state (mostly for testing)
)
{
    TPM_RC               retVal = TPM_RC_SUCCESS;
    TPM2B_PUBLIC_KEY_RSA dataIn;
    //
    // if the input and output buffers are the same, copy the input to a scratch
    // buffer so that things don't get messed up.
    if(dIn == &cOut->b)
    {
        MemoryCopy2B(&dataIn.b, dIn, sizeof(dataIn.t.buffer));
        dIn = &dataIn.b;
    }
    // All encryption schemes return the same size of data
    cOut->t.size = key->publicArea.unique.rsa.t.size;
    TPM_DO_SELF_TEST(scheme->scheme);

    switch(scheme->scheme)
    {
        case TPM_ALG_NULL:  // 'raw' encryption
        {
            INT32 i;
            INT32 dSize = dIn->size;
            // dIn can have more bytes than cOut as long as the extra bytes
            // are zero. Note: the more significant bytes of a number in a byte
            // buffer are the bytes at the start of the array.
            for(i = 0; (i < dSize) && (dIn->buffer[i] == 0); i++)
                ;
            dSize -= i;
            if(dSize > cOut->t.size)
                ERROR_EXIT(TPM_RC_VALUE);
            // Pad cOut with zeros if dIn is smaller
            memset(cOut->t.buffer, 0, cOut->t.size - dSize);
            // And copy the rest of the value
            memcpy(&cOut->t.buffer[cOut->t.size - dSize], &dIn->buffer[i], dSize);

            // If the size of dIn is the same as cOut dIn could be larger than
            // the modulus. If it is, then RSAEP() will catch it.
        }
        break;
        case TPM_ALG_RSAES:
            retVal = RSAES_PKCS1v1_5Encode(&cOut->b, dIn, rand);
            break;
```

```
        case TPM_ALG_OAEP:
            retVal =
                OaepEncode(&cOut->b, scheme->details.oaep.hashAlg, label, dIn, rand);
            break;
        default:
            ERROR_EXIT(TPM_RC_SCHEME);
            break;
    }
    // All the schemes that do padding will come here for the encryption step
    // Check that the Encoding worked
    if(retVal == TPM_RC_SUCCESS)
        // Padding OK so do the encryption
        retVal = RSAEP(&cOut->b, key);
Exit:
    return retVal;
}

//*** CryptRsaDecrypt()
// This is the entry point for decryption using RSA. Decryption is
// use of the private exponent. The 'padType' parameter determines what
// padding was used.
//
//  Return Type: TPM_RC
//      TPM_RC_SIZE         'cInSize' is not the same as the size of the public
//                           modulus of 'key'; or numeric value of the encrypted
//                           data is greater than the modulus
//      TPM_RC_VALUE        'dOutSize' is not large enough for the result
//      TPM_RC_SCHEME       'padType' is not supported
//
LIB_EXPORT TPM_RC CryptRsaDecrypt(
    TPM2B*             dOut,    // OUT: the decrypted data
    TPM2B*             cIn,     // IN: the data to decrypt
    OBJECT*            key,     // IN: the key to use for decryption
    TPMT_RSA_DECRYPT* scheme,   // IN: the padding scheme
    const TPM2B*       label    // IN: in case it is needed for the scheme
)
{
    TPM_RC retVal;

    // Make sure that the necessary parameters are provided
    pAssert(cIn != NULL && dOut != NULL && key != NULL);

    // Size is checked to make sure that the encrypted value is the right size
    if(cIn->size != key->publicArea.unique.rsa.t.size)
        ERROR_EXIT(TPM_RC_SIZE);

    TPM_DO_SELF_TEST(scheme->scheme);

    // For others that do padding, do the decryption in place and then
    // go handle the decoding.
    retVal = RSADP(cIn, key);
    if(retVal == TPM_RC_SUCCESS)
    {
        // Remove padding
        switch(scheme->scheme)
        {
            case TPM_ALG_NULL:
                if(dOut->size < cIn->size)
                    return TPM_RC_VALUE;
                MemoryCopy2B(dOut, cIn, dOut->size);
                break;
            case TPM_ALG_RSAES:
                retVal = RSAES_Decode(dOut, cIn);
                break;
            case TPM_ALG_OAEP:
                retVal = OaepDecode(dOut, scheme->details.oaep.hashAlg, label, cIn);
```

```
                break;
            default:
                retVal = TPM_RC_SCHEME;
                break;
        }
    }
Exit:
    return retVal;
}

//*** CryptRsaSign()
// This function is used to generate an RSA signature of the type indicated in
// 'scheme'.
//
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       'scheme' or 'hashAlg' are not supported
//      TPM_RC_VALUE        'hInSize' does not match 'hashAlg' (for RSASSA)
//
LIB_EXPORT TPM_RC CryptRsaSign(TPMT_SIGNATURE* sigOut,
                               OBJECT*         key,  // IN: key to use
                               TPM2B_DIGEST*   hIn,  // IN: the digest to sign
                               RAND_STATE* rand  // IN: the random number generator
                                                 //      to use (mostly for testing)
)
{
    TPM_RC retVal = TPM_RC_SUCCESS;
    UINT16 modSize;

    // parameter checks
    pAssert(sigOut != NULL && key != NULL && hIn != NULL);

    modSize = key->publicArea.unique.rsa.t.size;

    // for all non-null signatures, the size is the size of the key modulus
    sigOut->signature.rsapss.sig.t.size = modSize;

    TPM_DO_SELF_TEST(sigOut->sigAlg);

    switch(sigOut->sigAlg)
    {
        case TPM_ALG_NULL:
            sigOut->signature.rsapss.sig.t.size = 0;
            return TPM_RC_SUCCESS;
        case TPM_ALG_RSAPSS:
            retVal = PssEncode(&sigOut->signature.rsapss.sig.b,
                               sigOut->signature.rsapss.hash,
                               &hIn->b,
                               rand);
            break;
        case TPM_ALG_RSASSA:
            retVal = RSASSA_Encode(&sigOut->signature.rsassa.sig.b,
                                   sigOut->signature.rsassa.hash,
                                   &hIn->b);
            break;
        default:
            retVal = TPM_RC_SCHEME;
    }
    if(retVal == TPM_RC_SUCCESS)
    {
        // Do the encryption using the private key
        retVal = RSADP(&sigOut->signature.rsapss.sig.b, key);
    }
    return retVal;
}

//*** CryptRsaValidateSignature()
```

```
// This function is used to validate an RSA signature. If the signature is valid
// TPM_RC_SUCCESS is returned. If the signature is not valid, TPM_RC_SIGNATURE is
// returned. Other return codes indicate either parameter problems or fatal errors.
//
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE    the signature does not check
//      TPM_RC_SCHEME       unsupported scheme or hash algorithm
//
LIB_EXPORT TPM_RC CryptRsaValidateSignature(
    TPMT_SIGNATURE* sig,    // IN: signature
    OBJECT*         key,    // IN: public modulus
    TPM2B_DIGEST*   digest  // IN: The digest being validated
)
{
    TPM_RC retVal;
    //
    // Fatal programming errors
    pAssert(key != NULL && sig != NULL && digest != NULL);
    switch(sig->sigAlg)
    {
        case TPM_ALG_RSAPSS:
        case TPM_ALG_RSASSA:
            break;
        default:
            return TPM_RC_SCHEME;
    }

    // Errors that might be caused by calling parameters
    if(sig->signature.rsassa.sig.t.size != key->publicArea.unique.rsa.t.size)
        ERROR_EXIT(TPM_RC_SIGNATURE);

    TPM_DO_SELF_TEST(sig->sigAlg);

    // Decrypt the block
    retVal = RSAEP(&sig->signature.rsassa.sig.b, key);
    if(retVal == TPM_RC_SUCCESS)
    {
        switch(sig->sigAlg)
        {
            case TPM_ALG_RSAPSS:
                retVal = PssDecode(sig->signature.any.hashAlg,
                                   &digest->b,
                                   &sig->signature.rsassa.sig.b);
                break;
            case TPM_ALG_RSASSA:
                retVal = RSASSA_Decode(sig->signature.any.hashAlg,
                                       &digest->b,
                                       &sig->signature.rsassa.sig.b);
                break;
            default:
                return TPM_RC_SCHEME;
        }
    }
Exit:
    return (retVal != TPM_RC_SUCCESS) ? TPM_RC_SIGNATURE : TPM_RC_SUCCESS;
}

#  if SIMULATION && USE_RSA_KEY_CACHE
extern int s_rsaKeyCacheEnabled;
int        GetCachedRsaKey(
           TPMT_PUBLIC* publicArea, TPMT_SENSITIVE* sensitive, RAND_STATE* rand);
#    define GET_CACHED_KEY(publicArea, sensitive, rand) \
        (s_rsaKeyCacheEnabled && GetCachedRsaKey(publicArea, sensitive, rand))
#  else
#    define GET_CACHED_KEY(key, rand)
#  endif
```

```
//*** CryptRsaGenerateKey()
// Generate an RSA key from a provided seed
/*(See part 1 specification)
//  The formulation is:
//      KDFa(hash, seed, label, Name, Counter, bits)
//  Where:
//      hash        the nameAlg from the public template
//      seed        a seed (will be a primary seed for a primary key)
//      label       a distinguishing label including vendor ID and
//                  vendor-assigned part number for the TPM.
//      Name        the nameAlg from the template and the hash of the template
//                  using nameAlg.
//      Counter     a 32-bit integer that is incremented each time the KDF is
//                  called in order to produce a specific key. This value
//                  can be a 32-bit integer in host format and does not need
//                  to be put in canonical form.
//      bits        the number of bits needed for the key.
//  The following process is implemented to find a RSA key pair:
//  1. pick a random number with enough bits from KDFa as a prime candidate
//  2. set the first two significant bits and the least significant bit of the
//     prime candidate
//  3. check if the number is a prime. if not, pick another random number
//  4. Make sure the difference between the two primes are more than 2^104.
//     Otherwise, restart the process for the second prime
//  5. If the counter has reached its maximum but we still can not find a valid
//     RSA key pair, return an internal error. This is an artificial bound.
//     Other implementation may choose a smaller number to indicate how many
//     times they are willing to try.
*/
//  Return Type: TPM_RC
//      TPM_RC_CANCELED     operation was canceled
//      TPM_RC_RANGE        public exponent is not supported
//      TPM_RC_VALUE        could not find a prime using the provided parameters
LIB_EXPORT TPM_RC CryptRsaGenerateKey(
    TPMT_PUBLIC*    publicArea,
    TPMT_SENSITIVE* sensitive,
    RAND_STATE*     rand  // IN: if not NULL, the deterministic
                          //     RNG state
)
{
    UINT32 i;
    CRYPT_RSA_VAR(bnD);
    CRYPT_RSA_VAR(bnN);
    CRYPT_INT_WORD(bnPubExp);
    UINT32 e = publicArea->parameters.rsaDetail.exponent;
    int    keySizeInBits;
    TPM_RC retVal = TPM_RC_NO_RESULT;
    NEW_PRIVATE_EXPONENT(Z);
    //

    // Need to make sure that the caller did not specify an exponent that is
    // not supported
    e = publicArea->parameters.rsaDetail.exponent;
    if(e == 0)
        e = RSA_DEFAULT_PUBLIC_EXPONENT;
    else
    {
        if(e < 65537)
            ERROR_EXIT(TPM_RC_RANGE);
        // Check that e is prime
        if(!IsPrimeInt(e))
            ERROR_EXIT(TPM_RC_RANGE);
    }
    ExtMath_SetWord(bnPubExp, e);
```

```
    // check for supported key size.
    keySizeInBits = publicArea->parameters.rsaDetail.keyBits;
    if(((keySizeInBits % 1024) != 0)
       || (keySizeInBits > MAX_RSA_KEY_BITS)  // this might be redundant, but...
       || (keySizeInBits == 0))
        ERROR_EXIT(TPM_RC_VALUE);

    // Set the prime size for instrumentation purposes
    INSTRUMENT_SET(PrimeIndex, PRIME_INDEX(keySizeInBits / 2));

#   if SIMULATION && USE_RSA_KEY_CACHE
    if(GET_CACHED_KEY(publicArea, sensitive, rand))
        return TPM_RC_SUCCESS;
#   endif

    // Make sure that key generation has been tested
    TPM_DO_SELF_TEST(TPM_ALG_NULL);

    // The prime is computed in P. When a new prime is found, Q is checked to
    // see if it is zero.  If so, P is copied to Q and a new P is found.
    // When both P and Q are non-zero, the modulus and
    // private exponent are computed and a trial encryption/decryption is
    // performed.  If the encrypt/decrypt fails, assume that at least one of the
    // primes is composite. Since we don't know which one, set Q to zero and start
    // over and find a new pair of primes.

    for(i = 1; (retVal == TPM_RC_NO_RESULT) && (i != 100); i++)
    {
        if(_plat__IsCanceled())
            ERROR_EXIT(TPM_RC_CANCELED);

        if(TpmRsa_GeneratePrimeForRSA(Z->P, keySizeInBits / 2, e, rand)
           == TPM_RC_FAILURE)
        {
            retVal = TPM_RC_FAILURE;
            goto Exit;
        }

        INSTRUMENT_INC(PrimeCounts[PrimeIndex]);

        // If this is the second prime, make sure that it differs from the
        // first prime by at least 2^100
        if(ExtMath_IsZero(Z->Q))
        {
            // copy p to q and compute another prime in p
            ExtMath_Copy(Z->Q, Z->P);
            continue;
        }
        // Make sure that the difference is at least 100 bits. Need to do it this
        // way because the big numbers are only positive values
        if(ExtMath_UnsignedCmp(Z->P, Z->Q) < 0)
            ExtMath_Subtract(bnD, Z->Q, Z->P);
        else
            ExtMath_Subtract(bnD, Z->P, Z->Q);
        if(ExtMath_MostSigBitNum(bnD) < 100)
            continue;

        //Form the public modulus and set the unique value
        ExtMath_Multiply(bnN, Z->P, Z->Q);
        TpmMath_IntTo2B(
            bnN, &publicArea->unique.rsa.b, (NUMBYTES)BITS_TO_BYTES(keySizeInBits));
        // Make sure everything came out right. The MSb of the values must be one
        if(((publicArea->unique.rsa.t.buffer[0] & 0x80) == 0)
           || (publicArea->unique.rsa.t.size
               != (NUMBYTES)BITS_TO_BYTES(keySizeInBits)))
            FAIL(FATAL_ERROR_INTERNAL);
```

```
        // Make sure that we can form the private exponent values
        if(ComputePrivateExponent(bnPubExp, Z) != TRUE)
        {
            // If ComputePrivateExponent could not find an inverse for
            // Q, then copy P and recompute P. This might
            // cause both to be recomputed if P is also zero
            if(ExtMath_IsZero(Z->Q))
                ExtMath_Copy(Z->Q, Z->P);
            continue;
        }

        // Pack the private exponent into the sensitive area
        PackExponent(&sensitive->sensitive.rsa, Z);
        // Make sure everything came out right. The MSb of the values must be one
        if(((publicArea->unique.rsa.t.buffer[0] & 0x80) == 0)
           || ((sensitive->sensitive.rsa.t.buffer[0] & 0x80) == 0))
            FAIL(FATAL_ERROR_INTERNAL);

        retVal = TPM_RC_SUCCESS;
        // Do a trial encryption decryption if this is a signing key
        if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
        {
            CRYPT_RSA_VAR(temp1);
            CRYPT_RSA_VAR(temp2);
            TpmMath_GetRandomInRange(temp1, bnN, rand);

            // Encrypt with public exponent...
            ExtMath_ModExp(temp2, temp1, bnPubExp, bnN);
            // ...  then decrypt with private exponent
            RsaPrivateKeyOp(temp2, Z);

            // If the starting and ending values are not the same,
            // start over )-;
            if(ExtMath_UnsignedCmp(temp2, temp1) != 0)
            {
                ExtMath_SetWord(Z->Q, 0);
                retVal = TPM_RC_NO_RESULT;
            }
        }
    }
Exit:
    return retVal;
}

#endif  // ALG_RSA
```

7.148 /tpm/src/crypt/CryptSelfTest.c

```
//** Introduction
// The functions in this file are designed to support self-test of cryptographic
// functions in the TPM. The TPM allows the user to decide whether to run self-test
// on a demand basis or to run all the self-tests before proceeding.
//
// The self-tests are controlled by a set of bit vectors. The
// 'g_untestedDecryptionAlgorithms' vector has a bit for each decryption algorithm
// that needs to be tested and 'g_untestedEncryptionAlgorithms' has a bit for
// each encryption algorithm that needs to be tested. Before an algorithm
// is used, the appropriate vector is checked (indexed using the algorithm ID).
// If the bit is 1, then the test function should be called.
//
// For more information, see TpmSelfTests.txt

#include "Tpm.h"
```

```
//** Functions

//*** RunSelfTest()
// Local function to run self-test
static TPM_RC CryptRunSelfTests(
    ALGORITHM_VECTOR* toTest  // IN: the vector of the algorithms to test
)
{
    TPM_ALG_ID alg;

    // For each of the algorithms that are in the toTestVecor, need to run a
    // test
    for(alg = TPM_ALG_FIRST; alg <= TPM_ALG_LAST; alg++)
    {
        if(TEST_BIT(alg, *toTest))
        {
            TPM_RC result = CryptTestAlgorithm(alg, toTest);
            if(result != TPM_RC_SUCCESS)
                return result;
        }
    }
    return TPM_RC_SUCCESS;
}

//*** CryptSelfTest()
// This function is called to start/complete a full self-test.
// If 'fullTest' is NO, then only the untested algorithms will be run. If
// 'fullTest' is YES, then 'g_untestedDecryptionAlgorithms' is reinitialized and then
// all tests are run.
// This implementation of the reference design does not support processing outside
// the framework of a TPM command. As a consequence, this command does not
// complete until all tests are done. Since this can take a long time, the TPM
// will check after each test to see if the command is canceled. If so, then the
// TPM will returned TPM_RC_CANCELLED. To continue with the self-tests, call
// TPM2_SelfTest(fullTest == No) and the TPM will complete the testing.
//  Return Type: TPM_RC
//      TPM_RC_CANCELED         if the command is canceled
LIB_EXPORT
TPM_RC
CryptSelfTest(TPMI_YES_NO fullTest  // IN: if full test is required
)
{
#if ALLOW_FORCE_FAILURE_MODE
    if(g_forceFailureMode)
        FAIL(FATAL_ERROR_FORCED);
#endif

    // If the caller requested a full test, then reset the to test vector so that
    // all the tests will be run
    if(fullTest == YES)
    {
        MemoryCopy(g_toTest, g_implementedAlgorithms, sizeof(g_toTest));
    }
    return CryptRunSelfTests(&g_toTest);
}

//*** CryptIncrementalSelfTest()
// This function is used to perform an incremental self-test. This implementation
// will perform the toTest values before returning. That is, it assumes that the
// TPM cannot perform background tasks between commands.
//
// This command may be canceled. If it is, then there is no return result.
// However, this command can be run again and the incremental progress will not
// be lost.
//  Return Type: TPM_RC
//      TPM_RC_CANCELED         processing of this command was canceled
```

```
//      TPM_RC_TESTING          if toTest list is not empty
//      TPM_RC_VALUE            an algorithm in the toTest list is not implemented
TPM_RC
CryptIncrementalSelfTest(TPML_ALG* toTest,   // IN: list of algorithms to be tested
                         TPML_ALG* toDoList  // OUT: list of algorithms needing test
)
{
    ALGORITHM_VECTOR toTestVector = {0};
    TPM_ALG_ID       alg;
    UINT32           i;

    pAssert(toTest != NULL && toDoList != NULL);
    if(toTest->count > 0)
    {
        // Transcribe the toTest list into the toTestVector
        for(i = 0; i < toTest->count; i++)
        {
            alg = toTest->algorithms[i];

            // make sure that the algorithm value is not out of range
            if((alg > TPM_ALG_LAST) || !TEST_BIT(alg, g_implementedAlgorithms))
                return TPM_RC_VALUE;
            SET_BIT(alg, toTestVector);
        }
        // Run the test
        if(CryptRunSelfTests(&toTestVector) == TPM_RC_CANCELED)
            return TPM_RC_CANCELED;
    }
    // Fill in the toDoList with the algorithms that are still untested
    toDoList->count = 0;

    for(alg = TPM_ALG_FIRST;
        toDoList->count < MAX_ALG_LIST_SIZE && alg <= TPM_ALG_LAST;
        alg++)
    {
        if(TEST_BIT(alg, g_toTest))
            toDoList->algorithms[toDoList->count++] = alg;
    }
    return TPM_RC_SUCCESS;
}

//*** CryptInitializeToTest()
// This function will initialize the data structures for testing all the
// algorithms. This should not be called unless CryptAlgsSetImplemented() has
// been called
void CryptInitializeToTest(void)
{
    // Indicate that nothing has been tested
    memset(&g_cryptoSelfTestState, 0, sizeof(g_cryptoSelfTestState));

    // Copy the implemented algorithm vector
    MemoryCopy(g_toTest, g_implementedAlgorithms, sizeof(g_toTest));

    // Setting the algorithm to null causes the test function to just clear
    // out any algorithms for which there is no test.
    CryptTestAlgorithm(TPM_ALG_ERROR, &g_toTest);

    return;
}

//*** CryptTestAlgorithm()
// Only point of contact with the actual self tests. If a self-test fails, there
// is no return and the TPM goes into failure mode.
// The call to TestAlgorithm uses an algorithm selector and a bit vector. When the
// test is run, the corresponding bit in 'toTest' and in 'g_toTest' is CLEAR. If
// 'toTest' is NULL, then only the bit in 'g_toTest' is CLEAR.
```

```
// There is a special case for the call to TestAlgorithm(). When 'alg' is
// ALG_ERROR, TestAlgorithm() will CLEAR any bit in 'toTest' for which it has
// no test. This allows the knowledge about which algorithms have test to be
// accessed through the interface that provides the test.
//  Return Type: TPM_RC
//      TPM_RC_CANCELED     test was canceled
LIB_EXPORT
TPM_RC
CryptTestAlgorithm(TPM_ALG_ID alg, ALGORITHM_VECTOR* toTest)
{
    TPM_RC result;
#if ENABLE_SELF_TESTS
    result = TestAlgorithm(alg, toTest);
#else
    // If this is an attempt to determine the algorithms for which there is a
    // self test, pretend that all of them do. We do that by not clearing any
    // of the algorithm bits. When/if this function is called to run tests, it
    // will over report. This can be changed so that any call to check on which
    // algorithms have tests, 'toTest' can be cleared.
    if(alg != TPM_ALG_ERROR)
    {
        CLEAR_BIT(alg, g_toTest);
        if(toTest != NULL)
            CLEAR_BIT(alg, *toTest);
    }
    result = TPM_RC_SUCCESS;
#endif
    return result;
}
```

7.149 /tpm/src/crypt/CryptSmac.c

```
//** Introduction
//
// This file contains the implementation of the message authentication codes based
// on a symmetric block cipher. These functions only use the single block
// encryption functions of the selected symmetric cryptographic library.

//** Includes, Defines, and Typedefs
#define _CRYPT_HASH_C_
#include "Tpm.h"

#if SMAC_IMPLEMENTED

//*** CryptSmacStart()
// Function to start an SMAC.
UINT16
CryptSmacStart(HASH_STATE*        state,
               TPMU_PUBLIC_PARMS* keyParameters,
               TPM_ALG_ID         macAlg,  // IN: the type of MAC
               TPM2B*             key)
{
    UINT16 retVal = 0;
    //
    // Make sure that the key size is correct. This should have been checked
    // at key load, but...
    if(BITS_TO_BYTES(keyParameters->symDetail.sym.keyBits.sym) == key->size)
    {
        switch(macAlg)
        {
#  if ALG_CMAC
            case TPM_ALG_CMAC:
                retVal =
                    CryptCmacStart(&state->state.smac, keyParameters, macAlg, key);
                break;
```

```
#  endif
            default:
                break;
        }
    }
    state->type = (retVal != 0) ? HASH_STATE_SMAC : HASH_STATE_EMPTY;
    return retVal;
}

//*** CryptMacStart()
// Function to start either an HMAC or an SMAC. Cannot reuse the CryptHmacStart
// function because of the difference in number of parameters.
UINT16
CryptMacStart(HMAC_STATE*        state,
              TPMU_PUBLIC_PARMS* keyParameters,
              TPM_ALG_ID         macAlg,  // IN: the type of MAC
              TPM2B*             key)
{
    MemorySet(state, 0, sizeof(HMAC_STATE));
    if(CryptHashIsValidAlg(macAlg, FALSE))
    {
        return CryptHmacStart(state, macAlg, key->size, key->buffer);
    }
    else if(CryptSmacIsValidAlg(macAlg, FALSE))
    {
        return CryptSmacStart(&state->hashState, keyParameters, macAlg, key);
    }
    else
        return 0;
}

//*** CryptMacEnd()
// Dispatch to the MAC end function using a size and buffer pointer.
UINT16
CryptMacEnd(HMAC_STATE* state, UINT32 size, BYTE* buffer)
{
    UINT16 retVal = 0;
    if(state->hashState.type == HASH_STATE_SMAC)
        retVal = (state->hashState.state.smac.smacMethods.end)(
            &state->hashState.state.smac.state, size, buffer);
    else if(state->hashState.type == HASH_STATE_HMAC)
        retVal = CryptHmacEnd(state, size, buffer);
    state->hashState.type = HASH_STATE_EMPTY;
    return retVal;
}

//*** CryptMacEnd2B()
// Dispatch to the MAC end function using a 2B.
UINT16
CryptMacEnd2B(HMAC_STATE* state, TPM2B* data)
{
    return CryptMacEnd(state, data->size, data->buffer);
}
#endif  // SMAC_IMPLEMENTED
```

7.150 /tpm/src/crypt/CryptSym.c

```
//** Introduction
//
// This file contains the implementation of the symmetric block cipher modes
// allowed for a TPM. These functions only use the single block encryption functions
// of the selected symmetric crypto library.

//** Includes, Defines, and Typedefs
#include "Tpm.h"
```

```
#include "CryptSym.h"

#define KEY_BLOCK_SIZES(ALG, alg)                                          \
    static const INT16 alg##KeyBlockSizes[] = {ALG##_KEY_SIZES_BITS, \
                                               -1,                     \
                                               ALG##_BLOCK_SIZES};

FOR_EACH_SYM(KEY_BLOCK_SIZES)

//** Initialization and Data Access Functions
//
//*** CryptSymInit()
// This function is called to do _TPM_Init processing
BOOL CryptSymInit(void)
{
    return TRUE;
}

//*** CryptSymStartup()
// This function is called to do TPM2_Startup() processing
BOOL CryptSymStartup(void)
{
    return TRUE;
}

//*** CryptGetSymmetricBlockSize()
// This function returns the block size of the algorithm. The table of bit sizes has
// an entry for each allowed key size. The entry for a key size is 0 if the TPM does
// not implement that key size. The key size table is delimited with a negative number
// (-1). After the delimiter is a list of block sizes with each entry corresponding
// to the key bit size. For most symmetric algorithms, the block size is the same
// regardless of the key size but this arrangement allows them to be different.
//  Return Type: INT16
//   <= 0     cipher not supported
//   > 0      the cipher block size in bytes
LIB_EXPORT INT16 CryptGetSymmetricBlockSize(
    TPM_ALG_ID symmetricAlg,  // IN: the symmetric algorithm
    UINT16     keySizeInBits  // IN: the key size
)
{
    const INT16* sizes;
    INT16        i;
#define ALG_CASE(SYM, sym)          \
    case TPM_ALG_##SYM:             \
        sizes = sym##KeyBlockSizes; \
        break
    switch(symmetricAlg)
    {
#define GET_KEY_BLOCK_POINTER(SYM, sym) \
    case TPM_ALG_##SYM:                 \
        sizes = sym##KeyBlockSizes;     \
        break;
        // Get the pointer to the block size array
        FOR_EACH_SYM(GET_KEY_BLOCK_POINTER);

        default:
            return 0;
    }
    // Find the index of the indicated keySizeInBits
    for(i = 0; *sizes >= 0; i++, sizes++)
    {
        if(*sizes == keySizeInBits)
            break;
    }
    // If sizes is pointing at the end of the list of key sizes, then the desired
```

```
    // key size was not found so set the block size to zero.
    if(*sizes++ < 0)
        return 0;
    // Advance until the end of the list is found
    while(*sizes++ >= 0)
        ;
    // sizes is pointing to the first entry in the list of block sizes. Use the
    // ith index to find the block size for the corresponding key size.
    return sizes[i];
}

//** Symmetric Encryption
// This function performs symmetric encryption based on the mode.
//  Return Type: TPM_RC
//      TPM_RC_SIZE         'dSize' is not a multiple of the block size for an
//                          algorithm that requires it
//      TPM_RC_FAILURE      Fatal error
LIB_EXPORT TPM_RC CryptSymmetricEncrypt(
    BYTE*       dOut,           // OUT:
    TPM_ALG_ID  algorithm,      // IN: the symmetric algorithm
    UINT16      keySizeInBits,  // IN: key size in bits
    const BYTE* key,            // IN: key buffer. The size of this buffer
                                //     in bytes is (keySizeInBits + 7) / 8
    TPM2B_IV*  ivInOut,         // IN/OUT: IV for decryption.
    TPM_ALG_ID mode,            // IN: Mode to use
    INT32      dSize,           // IN: data size (may need to be a
                                //     multiple of the blockSize)
    const BYTE* dIn             // IN: data buffer
)
{
    BYTE*                   pIv;
    int                     i;
    BYTE                    tmp[MAX_SYM_BLOCK_SIZE];
    BYTE*                   pT;
    tpmCryptKeySchedule_t   keySchedule;
    INT16                   blockSize;
    TpmCryptSetSymKeyCall_t encrypt;
    BYTE*                   iv;
    BYTE                    defaultIv[MAX_SYM_BLOCK_SIZE] = {0};
    //
    pAssert(dOut != NULL && key != NULL && dIn != NULL);
    if(dSize == 0)
        return TPM_RC_SUCCESS;

    TPM_DO_SELF_TEST(algorithm);
    blockSize = CryptGetSymmetricBlockSize(algorithm, keySizeInBits);
    if(blockSize == 0)
        return TPM_RC_FAILURE;
    // If the iv is provided, then it is expected to be block sized. In some cases,
    // the caller is providing an array of 0's that is equal to [MAX_SYM_BLOCK_SIZE]
    // with no knowledge of the actual block size. This function will set it.
    if((ivInOut != NULL) && (mode != TPM_ALG_ECB))
    {
        ivInOut->t.size = blockSize;
        iv              = ivInOut->t.buffer;
    }
    else
        iv = defaultIv;
    pIv = iv;

    // Create encrypt key schedule and set the encryption function pointer.
    switch(algorithm)
    {
        FOR_EACH_SYM(ENCRYPT_CASE)

        default:
```

```
            return TPM_RC_SYMMETRIC;
    }
    switch(mode)
    {
#if ALG_CTR
        case TPM_ALG_CTR:
            for(; dSize > 0; dSize -= blockSize)
            {
                // Encrypt the current value of the IV(counter)
                ENCRYPT(&keySchedule, iv, tmp);

                //increment the counter (counter is big-endian so start at end)
                for(i = blockSize - 1; i >= 0; i--)
                    if((iv[i] += 1) != 0)
                        break;
                // XOR the encrypted counter value with input and put into output
                pT = tmp;
                for(i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
                    *dOut++ = *dIn++ ^ *pT++;
            }
            break;
#endif
#if ALG_OFB
        case TPM_ALG_OFB:
            // This is written so that dIn and dOut may be the same
            for(; dSize > 0; dSize -= blockSize)
            {
                // Encrypt the current value of the "IV"
                ENCRYPT(&keySchedule, iv, iv);

                // XOR the encrypted IV into dIn to create the cipher text (dOut)
                pIv = iv;
                for(i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
                    *dOut++ = (*pIv++ ^ *dIn++);
            }
            break;
#endif
#if ALG_CBC
        case TPM_ALG_CBC:
            // For CBC the data size must be an even multiple of the
            // cipher block size
            if((dSize % blockSize) != 0)
                return TPM_RC_SIZE;
            // XOR the data block into the IV, encrypt the IV into the IV
            // and then copy the IV to the output
            for(; dSize > 0; dSize -= blockSize)
            {
                pIv = iv;
                for(i = blockSize; i > 0; i--)
                    *pIv++ ^= *dIn++;
                ENCRYPT(&keySchedule, iv, iv);
                pIv = iv;
                for(i = blockSize; i > 0; i--)
                    *dOut++ = *pIv++;
            }
            break;
#endif
        // CFB is not optional
        case TPM_ALG_CFB:
            // Encrypt the IV into the IV, XOR in the data, and copy to output
            for(; dSize > 0; dSize -= blockSize)
            {
                // Encrypt the current value of the IV
                ENCRYPT(&keySchedule, iv, iv);
                pIv = iv;
                for(i = (int)(dSize < blockSize) ? dSize : blockSize; i > 0; i--)
```

```
                    // XOR the data into the IV to create the cipher text
                    // and put into the output
                    *dOut++ = *pIv++ ^= *dIn++;
                }
                // If the inner loop (i loop) was smaller than blockSize, then dSize
                // would have been smaller than blockSize and it is now negative. If
                // it is negative, then it indicates how many bytes are needed to pad
                // out the IV for the next round.
                for(; dSize < 0; dSize++)
                    *pIv++ = 0;
                break;
#if ALG_ECB
            case TPM_ALG_ECB:
                // For ECB the data size must be an even multiple of the
                // cipher block size
                if((dSize % blockSize) != 0)
                    return TPM_RC_SIZE;
                // Encrypt the input block to the output block
                for(; dSize > 0; dSize -= blockSize)
                {
                    ENCRYPT(&keySchedule, dIn, dOut);
                    dIn  = &dIn[blockSize];
                    dOut = &dOut[blockSize];
                }
                break;
#endif
            default:
                return TPM_RC_FAILURE;
        }
        return TPM_RC_SUCCESS;
}

//*** CryptSymmetricDecrypt()
// This function performs symmetric decryption based on the mode.
//  Return Type: TPM_RC
//      TPM_RC_FAILURE      A fatal error
//      TPM_RCS_SIZE        'dSize' is not a multiple of the block size for an
//                          algorithm that requires it
LIB_EXPORT TPM_RC CryptSymmetricDecrypt(
    BYTE*       dOut,           // OUT: decrypted data
    TPM_ALG_ID  algorithm,      // IN: the symmetric algorithm
    UINT16      keySizeInBits,  // IN: key size in bits
    const BYTE* key,            // IN: key buffer. The size of this buffer
                                //     in bytes is (keySizeInBits + 7) / 8
    TPM2B_IV*  ivInOut,         // IN/OUT: IV for decryption.
    TPM_ALG_ID mode,            // IN: Mode to use
    INT32      dSize,           // IN: data size (may need to be a
                                //     multiple of the blockSize)
    const BYTE* dIn             // IN: data buffer
)
{
    BYTE*                   pIv;
    int                     i;
    BYTE                    tmp[MAX_SYM_BLOCK_SIZE];
    BYTE*                   pT;
    tpmCryptKeySchedule_t   keySchedule;
    INT16                   blockSize;
    BYTE*                   iv;
    TpmCryptSetSymKeyCall_t encrypt;
    TpmCryptSetSymKeyCall_t decrypt;
    BYTE                    defaultIv[MAX_SYM_BLOCK_SIZE] = {0};

    // These are used but the compiler can't tell because they are initialized
    // in case statements and it can't tell if they are always initialized
    // when needed, so... Comment these out if the compiler can tell or doesn't
    // care that these are initialized before use.
```

```
    encrypt = NULL;
    decrypt = NULL;

    pAssert(dOut != NULL && key != NULL && dIn != NULL);
    if(dSize == 0)
        return TPM_RC_SUCCESS;

    TPM_DO_SELF_TEST(algorithm);
    blockSize = CryptGetSymmetricBlockSize(algorithm, keySizeInBits);
    if(blockSize == 0)
        return TPM_RC_FAILURE;
    // If the iv is provided, then it is expected to be block sized. In some cases,
    // the caller is providing an array of 0's that is equal to [MAX_SYM_BLOCK_SIZE]
    // with no knowledge of the actual block size. This function will set it.
    if((ivInOut != NULL) && (mode != TPM_ALG_ECB))
    {
        ivInOut->t.size = blockSize;
        iv              = ivInOut->t.buffer;
    }
    else
        iv = defaultIv;

    pIv = iv;
    // Use the mode to select the key schedule to create. Encrypt always uses the
    // encryption schedule. Depending on the mode, decryption might use either
    // the decryption or encryption schedule.
    switch(mode)
    {
#if ALG_CBC || ALG_ECB
        case TPM_ALG_CBC:  // decrypt = decrypt
        case TPM_ALG_ECB:
            // For ECB and CBC, the data size must be an even multiple of the
            // cipher block size
            if((dSize % blockSize) != 0)
                return TPM_RC_SIZE;
            switch(algorithm)
            {
                FOR_EACH_SYM(DECRYPT_CASE)
                default:
                    return TPM_RC_SYMMETRIC;
            }
            break;
#endif
        default:
            // For the remaining stream ciphers, use encryption to decrypt
            switch(algorithm)
            {
                FOR_EACH_SYM(ENCRYPT_CASE)
                default:
                    return TPM_RC_SYMMETRIC;
            }
    }
    // Now do the mode-dependent decryption
    switch(mode)
    {
#if ALG_CBC
        case TPM_ALG_CBC:
            // Copy the input data to a temp buffer, decrypt the buffer into the
            // output, XOR in the IV, and copy the temp buffer to the IV and repeat.
            for(; dSize > 0; dSize -= blockSize)
            {
                pT = tmp;
                for(i = blockSize; i > 0; i--)
                    *pT++ = *dIn++;
                DECRYPT(&keySchedule, tmp, dOut);
                pIv = iv;
```

```
                pT  = tmp;
                for(i = blockSize; i > 0; i--)
                {
                    *dOut++ ^= *pIv;
                    *pIv++ = *pT++;
                }
            }
            break;
#endif
        case TPM_ALG_CFB:
            for(; dSize > 0; dSize -= blockSize)
            {
                // Encrypt the IV into the temp buffer
                ENCRYPT(&keySchedule, iv, tmp);
                pT  = tmp;
                pIv = iv;
                for(i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
                    // Copy the current cipher text to IV, XOR
                    // with the temp buffer and put into the output
                    *dOut++ = *pT++ ^ (*pIv++ = *dIn++);
            }
            // If the inner loop (i loop) was smaller than blockSize, then dSize
            // would have been smaller than blockSize and it is now negative
            // If it is negative, then it indicates how may fill bytes
            // are needed to pad out the IV for the next round.
            for(; dSize < 0; dSize++)
                *pIv++ = 0;

            break;
#if ALG_CTR
        case TPM_ALG_CTR:
            for(; dSize > 0; dSize -= blockSize)
            {
                // Encrypt the current value of the IV(counter)
                ENCRYPT(&keySchedule, iv, tmp);

                //increment the counter (counter is big-endian so start at end)
                for(i = blockSize - 1; i >= 0; i--)
                    if((iv[i] += 1) != 0)
                        break;
                // XOR the encrypted counter value with input and put into output
                pT = tmp;
                for(i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
                    *dOut++ = *dIn++ ^ *pT++;
            }
            break;
#endif
#if ALG_ECB
        case TPM_ALG_ECB:
            for(; dSize > 0; dSize -= blockSize)
            {
                DECRYPT(&keySchedule, dIn, dOut);
                dIn  = &dIn[blockSize];
                dOut = &dOut[blockSize];
            }
            break;
#endif
#if ALG_OFB
        case TPM_ALG_OFB:
            // This is written so that dIn and dOut may be the same
            for(; dSize > 0; dSize -= blockSize)
            {
                // Encrypt the current value of the "IV"
                ENCRYPT(&keySchedule, iv, iv);

                // XOR the encrypted IV into dIn to create the cipher text (dOut)
```

```
                pIv = iv;
                for(i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
                    *dOut++ = (*pIv++ ^ *dIn++);
            }
            break;
#endif
        default:
            return TPM_RC_FAILURE;
    }
    return TPM_RC_SUCCESS;
}

//*** CryptSymKeyValidate()
// Validate that a provided symmetric key meets the requirements of the TPM
//  Return Type: TPM_RC
//      TPM_RC_KEY_SIZE         Key size specifiers do not match
//      TPM_RC_KEY              Key is not allowed
TPM_RC
CryptSymKeyValidate(TPMT_SYM_DEF_OBJECT* symDef, TPM2B_SYM_KEY* key)
{
    if(key->t.size != BITS_TO_BYTES(symDef->keyBits.sym))
        return TPM_RCS_KEY_SIZE;
    return TPM_RC_SUCCESS;
}
```

7.151 /tpm/src/crypt/CryptUtil.c

```
//** Introduction
//
//  This module contains the interfaces to the CryptoEngine and provides
//  miscellaneous cryptographic functions in support of the TPM.
//

//** Includes
#include "Tpm.h"
#include "Marshal.h"

//**********************************************************************/
//**     Hash/HMAC Functions
//**********************************************************************/

//*** CryptHmacSign()
// Sign a digest using an HMAC key. This an HMAC of a digest, not an HMAC of a
// message.
//  Return Type: TPM_RC
//      TPM_RC_HASH         not a valid hash
static TPM_RC CryptHmacSign(TPMT_SIGNATURE* signature,  // OUT: signature
                            OBJECT*         signKey,    // IN: HMAC key sign the hash
                            TPM2B_DIGEST*   hashData    // IN: hash to be signed
)
{
    HMAC_STATE hmacState;
    UINT32     digestSize;

    digestSize = CryptHmacStart2B(&hmacState,
                                  signature->signature.any.hashAlg,
                                  &signKey->sensitive.sensitive.bits.b);
    CryptDigestUpdate2B(&hmacState.hashState, &hashData->b);
    CryptHmacEnd(&hmacState, digestSize, (BYTE*)&signature->signature.hmac.digest);
    return TPM_RC_SUCCESS;
}

//*** CryptHMACVerifySignature()
// This function will verify a signature signed by a HMAC key.
// Note that a caller needs to prepare 'signature' with the signature algorithm
```

```
// (TPM_ALG_HMAC) and the hash algorithm to use. This function then builds a
// signature of that type.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME           not the proper scheme for this key type
//      TPM_RC_SIGNATURE        if invalid input or signature is not genuine
static TPM_RC CryptHMACVerifySignature(
    OBJECT*         signKey,   // IN: HMAC key signed the hash
    TPM2B_DIGEST*   hashData,  // IN: digest being verified
    TPMT_SIGNATURE* signature  // IN: signature to be verified
)
{
    TPMT_SIGNATURE         test;
    TPMT_KEYEDHASH_SCHEME* keyScheme =
        &signKey->publicArea.parameters.keyedHashDetail.scheme;
    //
    if((signature->sigAlg != TPM_ALG_HMAC)
       || (signature->signature.hmac.hashAlg == TPM_ALG_NULL))
        return TPM_RC_SCHEME;
    // This check is not really needed for verification purposes. However, it does
    // prevent someone from trying to validate a signature using a weaker hash
    // algorithm than otherwise allowed by the key. That is, a key with a scheme
    // other than TMP_ALG_NULL can only be used to validate signatures that have
    // a matching scheme.
    if((keyScheme->scheme != TPM_ALG_NULL)
       && ((keyScheme->scheme != signature->sigAlg)
           || (keyScheme->details.hmac.hashAlg != signature->signature.any.hashAlg)))
        return TPM_RC_SIGNATURE;
    test.sigAlg                 = signature->sigAlg;
    test.signature.hmac.hashAlg = signature->signature.hmac.hashAlg;

    CryptHmacSign(&test, signKey, hashData);

    // Compare digest
    if(!MemoryEqual(&test.signature.hmac.digest,
                    &signature->signature.hmac.digest,
                    CryptHashGetDigestSize(signature->signature.any.hashAlg)))
        return TPM_RC_SIGNATURE;

    return TPM_RC_SUCCESS;
}

//*** CryptGenerateKeyedHash()
// This function creates a keyedHash object.
// Return type: TPM_RC
//      TPM_RC_NO_RESULT    cannot get values from random number generator
//      TPM_RC_SIZE         sensitive data size is larger than allowed for
//                          the scheme
static TPM_RC CryptGenerateKeyedHash(
    TPMT_PUBLIC* publicArea,                // IN/OUT: the public area template
                                            //     for the new key.
    TPMT_SENSITIVE*        sensitive,       // OUT: sensitive area
    TPMS_SENSITIVE_CREATE* sensitiveCreate, // IN: sensitive creation data
    RAND_STATE*            rand             // IN: "entropy" source
)
{
    TPMT_KEYEDHASH_SCHEME* scheme;
    TPM_ALG_ID             hashAlg;
    UINT16                 digestSize;

    scheme = &publicArea->parameters.keyedHashDetail.scheme;

    if(publicArea->type != TPM_ALG_KEYEDHASH)
        return TPM_RC_FAILURE;

    // Pick the limiting hash algorithm
    if(scheme->scheme == TPM_ALG_NULL)
```

```
        hashAlg = publicArea->nameAlg;
    else if(scheme->scheme == TPM_ALG_XOR)
        hashAlg = scheme->details.xor.hashAlg;
    else
        hashAlg = scheme->details.hmac.hashAlg;
    digestSize = CryptHashGetDigestSize(hashAlg);

    // if this is a signing or a decryption key, then the limit
    // for the data size is the block size of the hash. This limit
    // is set because larger values have lower entropy because of the
    // HMAC function. The lower limit is 1/2 the size of the digest
    //
    //If the user provided the key, check that it is a proper size
    if(sensitiveCreate->data.t.size != 0)
    {
        if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, decrypt)
           || IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
        {
            if(sensitiveCreate->data.t.size > CryptHashGetBlockSize(hashAlg))
                return TPM_RC_SIZE;
#if 0  // May make this a FIPS-mode requirement
            if(sensitiveCreate->data.t.size < (digestSize / 2))
                return TPM_RC_SIZE;
#endif
        }
        // If this is a data blob, then anything that will get past the unmarshaling
        // is OK
        MemoryCopy2B(&sensitive->sensitive.bits.b,
                     &sensitiveCreate->data.b,
                     sizeof(sensitive->sensitive.bits.t.buffer));
    }
    else
    {
        // The TPM is going to generate the data so set the size to be the
        // size of the digest of the algorithm
        sensitive->sensitive.bits.t.size =
            DRBG_Generate(rand, sensitive->sensitive.bits.t.buffer, digestSize);
        if(sensitive->sensitive.bits.t.size == 0)
            return (g_inFailureMode) ? TPM_RC_FAILURE : TPM_RC_NO_RESULT;
    }
    return TPM_RC_SUCCESS;
}

//*** CryptIsSchemeAnonymous()
// This function is used to test a scheme to see if it is an anonymous scheme
// The only anonymous scheme is ECDAA. ECDAA can be used to do things
// like U-Prove.
BOOL CryptIsSchemeAnonymous(TPM_ALG_ID scheme  // IN: the scheme algorithm to test
)
{
    return scheme == TPM_ALG_ECDAA;
}

//**** ********************************************************************
//** Symmetric Functions
//**** ********************************************************************

//*** ParmDecryptSym()
//  This function performs parameter decryption using symmetric block cipher.
/*(See Part 1 specification)
// Symmetric parameter decryption
//      When parameter decryption uses a symmetric block cipher, a decryption
//      key and IV will be generated from:
//      KDFa(hash, sessionAuth, "CFB", nonceNewer, nonceOlder, bits)    (24)
//      Where:
//      hash            the hash function associated with the session
```

```
//      sessionAuth     the sessionAuth associated with the session
//      nonceNewer      nonceCaller for a command
//      nonceOlder      nonceTPM for a command
//      bits            the number of bits required for the symmetric key
//                      plus an IV
*/
void ParmDecryptSym(TPM_ALG_ID symAlg,          // IN: the symmetric algorithm
                    TPM_ALG_ID hash,            // IN: hash algorithm for KDFa
                    UINT16     keySizeInBits,   // IN: the key size in bits
                    TPM2B*     key,             // IN: KDF HMAC key
                    TPM2B*     nonceCaller,     // IN: nonce caller
                    TPM2B*     nonceTpm,        // IN: nonce TPM
                    UINT32     dataSize,        // IN: size of parameter buffer
                    BYTE*      data             // OUT: buffer to be decrypted
)
{
    // KDF output buffer
    // It contains parameters for the CFB encryption
    // From MSB to LSB, they are the key and iv
    BYTE symParmString[MAX_SYM_KEY_BYTES + MAX_SYM_BLOCK_SIZE];
    // Symmetric key size in byte
    UINT16   keySize = (keySizeInBits + 7) / 8;
    TPM2B_IV iv;

    iv.t.size = CryptGetSymmetricBlockSize(symAlg, keySizeInBits);
    // If there is decryption to do...
    if(iv.t.size > 0)
    {
        // Generate key and iv
        CryptKDFa(hash,
                  key,
                  CFB_KEY,
                  nonceCaller,
                  nonceTpm,
                  keySizeInBits + (iv.t.size * 8),
                  symParmString,
                  NULL,
                  FALSE);
        MemoryCopy(iv.t.buffer, &symParmString[keySize], iv.t.size);

        CryptSymmetricDecrypt(data,
                              symAlg,
                              keySizeInBits,
                              symParmString,
                              &iv,
                              TPM_ALG_CFB,
                              dataSize,
                              data);
    }
    return;
}

//*** ParmEncryptSym()
//  This function performs parameter encryption using symmetric block cipher.
/*(See part 1 specification)
//      When parameter decryption uses a symmetric block cipher, an encryption
//      key and IV will be generated from:
//      KDFa(hash, sessionAuth, "CFB", nonceNewer, nonceOlder, bits)    (24)
//      Where:
//      hash            the hash function associated with the session
//      sessionAuth     the sessionAuth associated with the session
//      nonceNewer      nonceTPM for a response
//      nonceOlder      nonceCaller for a response
//      bits            the number of bits required for the symmetric key
//                      plus an IV
*/
```

```
void ParmEncryptSym(TPM_ALG_ID symAlg,          // IN: symmetric algorithm
                    TPM_ALG_ID hash,            // IN: hash algorithm for KDFa
                    UINT16     keySizeInBits,   // IN: symmetric key size in bits
                    TPM2B*     key,             // IN: KDF HMAC key
                    TPM2B*     nonceCaller,     // IN: nonce caller
                    TPM2B*     nonceTpm,        // IN: nonce TPM
                    UINT32     dataSize,        // IN: size of parameter buffer
                    BYTE*      data             // OUT: buffer to be encrypted
)
{
    // KDF output buffer
    // It contains parameters for the CFB encryption
    BYTE symParmString[MAX_SYM_KEY_BYTES + MAX_SYM_BLOCK_SIZE];

    // Symmetric key size in bytes
    UINT16   keySize = (keySizeInBits + 7) / 8;

    TPM2B_IV iv;

    iv.t.size = CryptGetSymmetricBlockSize(symAlg, keySizeInBits);
    // See if there is any encryption to do
    if(iv.t.size > 0)
    {
        // Generate key and iv
        CryptKDFa(hash,
                  key,
                  CFB_KEY,
                  nonceTpm,
                  nonceCaller,
                  keySizeInBits + (iv.t.size * 8),
                  symParmString,
                  NULL,
                  FALSE);
        MemoryCopy(iv.t.buffer, &symParmString[keySize], iv.t.size);

        CryptSymmetricEncrypt(data,
                              symAlg,
                              keySizeInBits,
                              symParmString,
                              &iv,
                              TPM_ALG_CFB,
                              dataSize,
                              data);
    }
    return;
}

//*** CryptGenerateKeySymmetric()
// This function generates a symmetric cipher key. The derivation process is
// determined by the type of the provided 'rand'
// Return type: TPM_RC
//      TPM_RC_NO_RESULT    cannot get a random value
//      TPM_RC_KEY_SIZE     key size in the public area does not match the size
//                          in the sensitive creation area
//      TPM_RC_KEY          provided key value is not allowed
static TPM_RC CryptGenerateKeySymmetric(
    TPMT_PUBLIC* publicArea,                // IN/OUT: The public area template
                                            //     for the new key.
    TPMT_SENSITIVE*        sensitive,       // OUT: sensitive area
    TPMS_SENSITIVE_CREATE* sensitiveCreate, // IN: sensitive creation data
    RAND_STATE*            rand             // IN: the "entropy" source for
)
{
    UINT16 keyBits = publicArea->parameters.symDetail.sym.keyBits.sym;
    TPM_RC result;
    //
```

```
    // only do multiples of RADIX_BITS
    if((keyBits % RADIX_BITS) != 0)
        return TPM_RC_KEY_SIZE;
    // If this is not a new key, then the provided key data must be the right size
    if(sensitiveCreate->data.t.size != 0)
    {
        result = CryptSymKeyValidate(&publicArea->parameters.symDetail.sym,
                                     (TPM2B_SYM_KEY*)&sensitiveCreate->data);
        if(result == TPM_RC_SUCCESS)
            MemoryCopy2B(&sensitive->sensitive.sym.b,
                         &sensitiveCreate->data.b,
                         sizeof(sensitive->sensitive.sym.t.buffer));
    }
    else
    {
        sensitive->sensitive.sym.t.size = DRBG_Generate(
            rand, sensitive->sensitive.sym.t.buffer, BITS_TO_BYTES(keyBits));
        if(g_inFailureMode)
            result = TPM_RC_FAILURE;
        else if(sensitive->sensitive.sym.t.size == 0)
            result = TPM_RC_NO_RESULT;
        else
            result = TPM_RC_SUCCESS;
    }
    return result;
}

//*** CryptXORObfuscation()
// This function implements XOR obfuscation. It should not be called if the
// hash algorithm is not implemented. The only return value from this function
// is TPM_RC_SUCCESS.
void CryptXORObfuscation(TPM_ALG_ID hash,       // IN: hash algorithm for KDF
                         TPM2B*     key,        // IN: KDF key
                         TPM2B*     contextU,   // IN: contextU
                         TPM2B*     contextV,   // IN: contextV
                         UINT32     dataSize,   // IN: size of data buffer
                         BYTE*      data        // IN/OUT: data to be XORed in place
)
{
    BYTE   mask[MAX_DIGEST_SIZE];  // Allocate a digest sized buffer
    BYTE*  pm;
    UINT32 i;
    UINT32 counter     = 0;
    UINT16 hLen        = CryptHashGetDigestSize(hash);
    UINT32 requestSize = dataSize * 8;
    INT32  remainBytes = (INT32)dataSize;

    pAssert((key != NULL) && (data != NULL) && (hLen != 0));

    // Call KDFa to generate XOR mask
    for(; remainBytes > 0; remainBytes -= hLen)
    {
        // Make a call to KDFa to get next iteration
        CryptKDFa(hash,
                  key,
                  XOR_KEY,
                  contextU,
                  contextV,
                  requestSize,
                  mask,
                  &counter,
                  TRUE);

        // XOR next piece of the data
        pm = mask;
        for(i = hLen < remainBytes ? hLen : remainBytes; i > 0; i--)
```

```
            *data++ ^= *pm++;
    }
    return;
}

//**********************************************************************
//** Initialization and shut down
//**********************************************************************

//*** CryptInit()
// This function is called when the TPM receives a _TPM_Init indication.
//
// NOTE: The hash algorithms do not have to be tested, they just need to be
// available. They have to be tested before the TPM can accept HMAC authorization
// or return any result that relies on a hash algorithm.
//  Return Type: BOOL
//      TRUE(1)         initializations succeeded
//      FALSE(0)        initialization failed and caller should place the TPM into
//                      Failure Mode
BOOL CryptInit(void)
{
    BOOL ok;
    // Initialize the vector of implemented algorithms
    AlgorithmGetImplementedVector(&g_implementedAlgorithms);

    // Indicate that all test are necessary
    CryptInitializeToTest();

    // Do any library initializations that are necessary. If any fails,
    // the caller should go into failure mode;
    ok = ExtMath_LibInit();
    ok = ok && CryptSymInit();
    ok = ok && CryptRandInit();
    ok = ok && CryptHashInit();
#if ALG_RSA
    ok = ok && CryptRsaInit();
#endif  // ALG_RSA
#if ALG_ECC
    ok = ok && CryptEccInit();
#endif  // ALG_ECC
    return ok;
}

//*** CryptStartup()
// This function is called by TPM2_Startup() to initialize the functions in
// this cryptographic library and in the provided CryptoLibrary. This function
// and CryptUtilInit() are both provided so that the implementation may move the
// initialization around to get the best interaction.
//  Return Type: BOOL
//      TRUE(1)         startup succeeded
//      FALSE(0)        startup failed and caller should place the TPM into
//                      Failure Mode
BOOL CryptStartup(STARTUP_TYPE type  // IN: the startup type
)
{
    BOOL OK;
    NOT_REFERENCED(type);

    OK = CryptSymStartup() && CryptRandStartup() && CryptHashStartup()
#if ALG_RSA
         && CryptRsaStartup()
#endif  // ALG_RSA
#if ALG_ECC
         && CryptEccStartup()
#endif  // ALG_ECC
        ;
```

```
#if ALG_ECC
    // Don't directly check for SU_RESET because that is the default
    if(OK && (type != SU_RESTART) && (type != SU_RESUME))
    {
        // If the shutdown was orderly, then the values recovered from NV will
        // be OK to use.
        // Get a new  random commit nonce
        gr.commitNonce.t.size = sizeof(gr.commitNonce.t.buffer);
        CryptRandomGenerate(gr.commitNonce.t.size, gr.commitNonce.t.buffer);
        // Reset the counter and commit array
        gr.commitCounter = 0;
        MemorySet(gr.commitArray, 0, sizeof(gr.commitArray));
    }
#endif  // ALG_ECC
    return OK;
}

//**********************************************************************
//** Algorithm-Independent Functions
//**********************************************************************
//*** Introduction
// These functions are used generically when a function of a general type
// (e.g., symmetric encryption) is required.  The functions will modify the
// parameters as required to interface to the indicated algorithms.
//
//*** CryptIsAsymAlgorithm()
// This function indicates if an algorithm is an asymmetric algorithm.
//  Return Type: BOOL
//      TRUE(1)         if it is an asymmetric algorithm
//      FALSE(0)        if it is not an asymmetric algorithm
BOOL CryptIsAsymAlgorithm(TPM_ALG_ID algID  // IN: algorithm ID
)
{
    switch(algID)
    {
#if ALG_RSA
        case TPM_ALG_RSA:
#endif
#if ALG_ECC
        case TPM_ALG_ECC:
#endif
            return TRUE;
            break;
        default:
            break;
    }
    return FALSE;
}

//*** CryptSecretEncrypt()
// This function creates a secret value and its associated secret structure using
// an asymmetric algorithm.
//
// This function is used by TPM2_Rewrap() TPM2_MakeCredential(),
// and TPM2_Duplicate().
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES   'keyHandle' does not reference a valid decryption key
//      TPM_RC_KEY          invalid ECC key (public point is not on the curve)
//      TPM_RC_SCHEME       RSA key with an unsupported padding scheme
//      TPM_RC_VALUE        numeric value of the data to be decrypted is greater
//                          than the RSA key modulus
TPM_RC
CryptSecretEncrypt(OBJECT*      encryptKey,  // IN: encryption key object
                   const TPM2B* label,       // IN: a null-terminated string as L
                   TPM2B_DATA*  data,        // OUT: secret value
                   TPM2B_ENCRYPTED_SECRET* secret  // OUT: secret structure
```

```
)
{
    TPM_RC result = TPM_RC_SUCCESS;
    //
    if(data == NULL || secret == NULL)
        return TPM_RC_FAILURE;

    // CryptKDFe was fixed to not add a NULL byte as per NIST.SP.800-56Cr2.pdf
    // (required for ACVP tests). This check ensures backwards compatibility with
    // previous versions of the TPM reference code by verifying the label itself
    // has a NULL terminator.  Note the TPM spec specifies that the label must be NULL
    // terminated.  This is only a "new" failure path in the sense that it adds a
    // runtime check of hardcoded constants; provided the code is correct it will
never
    // fail, and running the compliance tests will verify this isn't hit.
    if((label == NULL) || (label->size == 0) || (label->buffer[label->size - 1] != 0))
        return TPM_RC_FAILURE;

    // The output secret value has the size of the digest produced by the nameAlg.
    data->t.size = CryptHashGetDigestSize(encryptKey->publicArea.nameAlg);

    if(!IS_ATTRIBUTE(encryptKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
        return TPM_RC_ATTRIBUTES;
    switch(encryptKey->publicArea.type)
    {
#if ALG_RSA
        case TPM_ALG_RSA:
        {
            // The encryption scheme is OAEP using the nameAlg of the encrypt key.
            TPMT_RSA_DECRYPT scheme;
            scheme.scheme                 = TPM_ALG_OAEP;
            scheme.details.anySig.hashAlg = encryptKey->publicArea.nameAlg;

            // Create secret data from RNG
            CryptRandomGenerate(data->t.size, data->t.buffer);

            // Encrypt the data by RSA OAEP into encrypted secret
            result = CryptRsaEncrypt((TPM2B_PUBLIC_KEY_RSA*)secret,
                                     &data->b,
                                     encryptKey,
                                     &scheme,
                                     label,
                                     NULL);
        }
        break;
#endif  // ALG_RSA

#if ALG_ECC
        case TPM_ALG_ECC:
        {
            TPMS_ECC_POINT      eccPublic;
            TPM2B_ECC_PARAMETER eccPrivate;
            TPMS_ECC_POINT      eccSecret;
            BYTE*               buffer = secret->t.secret;

            // Need to make sure that the public point of the key is on the
            // curve defined by the key.
            if(!CryptEccIsPointOnCurve(
                   encryptKey->publicArea.parameters.eccDetail.curveID,
                   &encryptKey->publicArea.unique.ecc))
                result = TPM_RC_KEY;
            else
            {
                // Call crypto engine to create an auxiliary ECC key
                // We assume crypt engine initialization should always success.
                // Otherwise, TPM should go to failure mode.
```

```
                CryptEccNewKeyPair(
                    &eccPublic,
                    &eccPrivate,
                    encryptKey->publicArea.parameters.eccDetail.curveID);
                // Marshal ECC public to secret structure. This will be used by the
                // recipient to decrypt the secret with their private key.
                secret->t.size = TPMS_ECC_POINT_Marshal(&eccPublic, &buffer, NULL);

                // Compute ECDH shared secret which is R = [d]Q where d is the
                // private part of the ephemeral key and Q is the public part of a
                // TPM key. TPM_RC_KEY error return from CryptComputeECDHSecret
                // because the auxiliary ECC key is just created according to the
                // parameters of input ECC encrypt key.
                if(CryptEccPointMultiply(
                       &eccSecret,
                       encryptKey->publicArea.parameters.eccDetail.curveID,
                       &encryptKey->publicArea.unique.ecc,
                       &eccPrivate,
                       NULL,
                       NULL)
                   != TPM_RC_SUCCESS)
                    result = TPM_RC_KEY;
                else
                {
                    // The secret value is computed from Z using KDFe as:
                    // secret := KDFe(HashID, Z, Use, PartyUInfo, PartyVInfo, bits)
                    // Where:
                    //  HashID  the nameAlg of the decrypt key
                    //  Z    the x coordinate (Px) of the product (P) of the point
                    //       (Q) of the secret and the private x coordinate (de,V)
                    //       of the decryption key
                    //  Use a null-terminated string containing "SECRET"
                    //  PartyUInfo  the x coordinate of the point in the secret
                    //              (Qe,U )
                    //  PartyVInfo  the x coordinate of the public key (Qs,V )
                    //  bits    the number of bits in the digest of HashID
                    // Retrieve seed from KDFe
                    CryptKDFe(encryptKey->publicArea.nameAlg,
                              &eccSecret.x.b,
                              label,
                              &eccPublic.x.b,
                              &encryptKey->publicArea.unique.ecc.x.b,
                              data->t.size * 8,
                              data->t.buffer);
                }
            }
        }
        break;
#endif  // ALG_ECC
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    return result;
}

//*** CryptSecretDecrypt()
// Decrypt a secret value by asymmetric (or symmetric) algorithm
// This function is used for ActivateCredential and Import for asymmetric
// decryption, and StartAuthSession for both asymmetric and symmetric
// decryption process
//
//  Return Type: TPM_RC
//      TPM_RC_ATTRIBUTES        RSA key is not a decryption key
//      TPM_RC_BINDING           Invalid RSA key (public and private parts are not
```

```
//                              cryptographically bound.
//      TPM_RC_ECC_POINT        ECC point in the secret is not on the curve
//      TPM_RC_INSUFFICIENT     failed to retrieve ECC point from the secret
//      TPM_RC_NO_RESULT        multiplication resulted in ECC point at infinity
//      TPM_RC_SIZE             data to decrypt is not of the same size as RSA key
//      TPM_RC_VALUE            For RSA key, numeric value of the encrypted data is
//                              greater than the modulus, or the recovered data is
//                              larger than the output buffer.
//                              For keyedHash or symmetric key, the secret is
//                              larger than the size of the digest produced by
//                              the name algorithm.
//      TPM_RC_FAILURE          internal error
TPM_RC
CryptSecretDecrypt(OBJECT*      decryptKey,   // IN: decrypt key
                   TPM2B_NONCE* nonceCaller,  // IN: nonceCaller.  It is needed for
                                              //     symmetric decryption.  For
                                              //     asymmetric decryption, this
                                              //     parameter is NULL
                   const TPM2B*            label,   // IN: a value for L
                   TPM2B_ENCRYPTED_SECRET* secret,  // IN: input secret
                   TPM2B_DATA*             data     // OUT: decrypted secret value
)
{
    TPM_RC result = TPM_RC_SUCCESS;

    // CryptKDFe was fixed to not add a NULL byte as per NIST.SP.800-56Cr2.pdf
    // (required for ACVP tests). This check ensures backwards compatibility with
    // previous versions of the TPM reference code by verifying the label itself
    // has a NULL terminator.  Note the TPM spec specifies that the label must be NULL
    // terminated.  This is only a "new" failure path in the sense that it adds a
    // runtime check of hardcoded constants; provided the code is correct it will
never
    // fail, and running the compliance tests will verify this isn't hit.
    if((label == NULL) || (label->size == 0) || (label->buffer[label->size - 1] != 0))
        return TPM_RC_FAILURE;

    // Decryption for secret
    switch(decryptKey->publicArea.type)
    {
#if ALG_RSA
        case TPM_ALG_RSA:
        {
            TPMT_RSA_DECRYPT scheme;
            TPMT_RSA_SCHEME* keyScheme =
                &decryptKey->publicArea.parameters.rsaDetail.scheme;
            UINT16 digestSize;

            scheme = *(TPMT_RSA_DECRYPT*)keyScheme;
            // If the key scheme is TPM_ALG_NULL, set the scheme to OAEP and
            // set the algorithm to the name algorithm.
            if(scheme.scheme == TPM_ALG_NULL)
            {
                // Use OAEP scheme
                scheme.scheme               = TPM_ALG_OAEP;
                scheme.details.oaep.hashAlg = decryptKey->publicArea.nameAlg;
            }
            // use the digestSize as an indicator of whether or not the scheme
            // is using a supported hash algorithm.
            // Note: depending on the scheme used for encryption, a hashAlg might
            // not be needed. However, the return value has to have some upper
            // limit on the size. In this case, it is the size of the digest of the
            // hash algorithm. It is checked after the decryption is done but, there
            // is no point in doing the decryption if the size is going to be
            // 'wrong' anyway.
            digestSize = CryptHashGetDigestSize(scheme.details.oaep.hashAlg);
            if(scheme.scheme != TPM_ALG_OAEP || digestSize == 0)
```

```
                return TPM_RC_SCHEME;

            // Set the output buffer capacity
            data->t.size = sizeof(data->t.buffer);

            // Decrypt seed by RSA OAEP
            result =
                CryptRsaDecrypt(&data->b, &secret->b, decryptKey, &scheme, label);
            if((result == TPM_RC_SUCCESS) && (data->t.size > digestSize))
                result = TPM_RC_VALUE;
        }
        break;
#endif  // ALG_RSA
#if ALG_ECC
        case TPM_ALG_ECC:
        {
            TPMS_ECC_POINT eccPublic;
            TPMS_ECC_POINT eccSecret;
            BYTE*          buffer = secret->t.secret;
            INT32          size   = secret->t.size;

            // Retrieve ECC point from secret buffer
            result = TPMS_ECC_POINT_Unmarshal(&eccPublic, &buffer, &size);
            if(result == TPM_RC_SUCCESS)
            {
                result = CryptEccPointMultiply(
                    &eccSecret,
                    decryptKey->publicArea.parameters.eccDetail.curveID,
                    &eccPublic,
                    &decryptKey->sensitive.sensitive.ecc,
                    NULL,
                    NULL);
                if(result == TPM_RC_SUCCESS)
                {
                    // Set the size of the "recovered" secret value to be the size
                    // of the digest produced by the nameAlg.
                    data->t.size =
                        CryptHashGetDigestSize(decryptKey->publicArea.nameAlg);

                    // The secret value is computed from Z using KDFe as:
                    // secret := KDFe(HashID, Z, Use, PartyUInfo, PartyVInfo, bits)
                    // Where:
                    //  HashID -- the nameAlg of the decrypt key
                    //  Z --  the x coordinate (Px) of the product (P) of the point
                    //        (Q) of the secret and the private x coordinate (de,V)
                    //        of the decryption key
                    //  Use -- a null-terminated string containing "SECRET"
                    //  PartyUInfo -- the x coordinate of the point in the secret
                    //             (Qe,U )
                    //  PartyVInfo -- the x coordinate of the public key (Qs,V )
                    //  bits -- the number of bits in the digest of HashID
                    // Retrieve seed from KDFe
                    CryptKDFe(decryptKey->publicArea.nameAlg,
                              &eccSecret.x.b,
                              label,
                              &eccPublic.x.b,
                              &decryptKey->publicArea.unique.ecc.x.b,
                              data->t.size * 8,
                              data->t.buffer);
                }
            }
        }
        break;
#endif  // ALG_ECC
#if !ALG_KEYEDHASH
#  error "KEYEDHASH support is required"
```

```
#endif
        case TPM_ALG_KEYEDHASH:
            // The seed size can not be bigger than the digest size of nameAlg
            if(secret->t.size
               > CryptHashGetDigestSize(decryptKey->publicArea.nameAlg))
                result = TPM_RC_VALUE;
            else
            {
                // Retrieve seed by XOR Obfuscation:
                //    seed = XOR(secret, hash, key, nonceCaller, nullNonce)
                //    where:
                //    secret  the secret parameter from the TPM2_StartAuthHMAC
                //            command that contains the seed value
                //    hash    nameAlg  of tpmKey
                //    key     the key or data value in the object referenced by
                //            entityHandle in the TPM2_StartAuthHMAC command
                //    nonceCaller the parameter from the TPM2_StartAuthHMAC command
                //    nullNonce   a zero-length nonce
                // XOR Obfuscation in place
                CryptXORObfuscation(decryptKey->publicArea.nameAlg,
                                    &decryptKey->sensitive.sensitive.bits.b,
                                    &nonceCaller->b,
                                    NULL,
                                    secret->t.size,
                                    secret->t.secret);
                // Copy decrypted seed
                MemoryCopy2B(&data->b, &secret->b, sizeof(data->t.buffer));
            }
            break;
        case TPM_ALG_SYMCIPHER:
        {
            TPM2B_IV              iv = {{0}};
            TPMT_SYM_DEF_OBJECT* symDef;
            // The seed size can not be bigger than the digest size of nameAlg
            if(secret->t.size
               > CryptHashGetDigestSize(decryptKey->publicArea.nameAlg))
                result = TPM_RC_VALUE;
            else
            {
                symDef    = &decryptKey->publicArea.parameters.symDetail.sym;
                iv.t.size = CryptGetSymmetricBlockSize(symDef->algorithm,
                                                       symDef->keyBits.sym);
                if(iv.t.size == 0)
                    return TPM_RC_FAILURE;
                if(nonceCaller->t.size >= iv.t.size)
                {
                    MemoryCopy(iv.t.buffer, nonceCaller->t.buffer, iv.t.size);
                }
                else
                {
                    if(nonceCaller->t.size > sizeof(iv.t.buffer))
                        return TPM_RC_FAILURE;
                    MemoryCopy(
                        iv.b.buffer, nonceCaller->t.buffer, nonceCaller->t.size);
                }
                // make sure secret will fit
                if(secret->t.size > sizeof(data->t.buffer))
                    return TPM_RC_FAILURE;
                data->t.size = secret->t.size;
                // CFB decrypt, using nonceCaller as iv
                CryptSymmetricDecrypt(data->t.buffer,
                                      symDef->algorithm,
                                      symDef->keyBits.sym,
                                      decryptKey->sensitive.sensitive.sym.t.buffer,
                                      &iv,
                                      TPM_ALG_CFB,
```

```
                                        secret->t.size,
                                        secret->t.secret);
            }
        }
        break;
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    return result;
}

//*** CryptParameterEncryption()
// This function does in-place encryption of a response parameter.
void CryptParameterEncryption(
    TPM_HANDLE handle,              // IN: encrypt session handle
    TPM2B*     nonceCaller,         // IN: nonce caller
    INT32      bufferSize,          // IN: size of parameter buffer
    UINT16     leadingSizeInByte,   // IN: the size of the leading size field in
                                    //     bytes
    TPM2B_AUTH* extraKey,           // IN: additional key material other than
                                    //     sessionAuth
    BYTE* buffer                    // IN/OUT: parameter buffer to be encrypted
)
{
    SESSION* session = SessionGet(handle);  // encrypt session
    TPM2B_TYPE(TEMP_KEY,
               (sizeof(extraKey->t.buffer) + sizeof(session->sessionKey.t.buffer)));
    TPM2B_TEMP_KEY key;              // encryption key
    UINT16         cipherSize = 0;  // size of cipher text

    if(bufferSize < leadingSizeInByte)
    {
        FAIL(FATAL_ERROR_INTERNAL);
        return;
    }

    // Parameter encryption for a non-2B is not supported.
    if(leadingSizeInByte != 2)
    {
        FAIL(FATAL_ERROR_INTERNAL);
        return;
    }

    // Retrieve encrypted data size.
    if(UINT16_Unmarshal(&cipherSize, &buffer, &bufferSize) != TPM_RC_SUCCESS)
    {
        FAIL(FATAL_ERROR_INTERNAL);
        return;
    }

    if(cipherSize > bufferSize)
    {
        FAIL(FATAL_ERROR_INTERNAL);
        return;
    }

    // Compute encryption key by concatenating sessionKey with extra key
    MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));
    MemoryConcat2B(&key.b, &extraKey->b, sizeof(key.t.buffer));

    if(session->symmetric.algorithm == TPM_ALG_XOR)

        // XOR parameter encryption formulation:
        //    XOR(parameter, hash, sessionAuth, nonceNewer, nonceOlder)
        CryptXORObfuscation(session->authHashAlg,
```

```
                            &(key.b),
                            &(session->nonceTPM.b),
                            nonceCaller,
                            (UINT32)cipherSize,
                            buffer);
    else
        ParmEncryptSym(session->symmetric.algorithm,
                       session->authHashAlg,
                       session->symmetric.keyBits.aes,
                       &(key.b),
                       nonceCaller,
                       &(session->nonceTPM.b),
                       (UINT32)cipherSize,
                       buffer);
    return;
}

//*** CryptParameterDecryption()
// This function does in-place decryption of a command parameter.
//  Return Type: TPM_RC
//      TPM_RC_SIZE             The number of bytes in the input buffer is less than
//                              the number of bytes to be decrypted.
TPM_RC
CryptParameterDecryption(
    TPM_HANDLE handle,              // IN: encrypted session handle
    TPM2B*     nonceCaller,         // IN: nonce caller
    INT32      bufferSize,          // IN: size of parameter buffer
    UINT16     leadingSizeInByte,   // IN: the size of the leading size field in
                                    //     byte
    TPM2B_AUTH* extraKey,           // IN: the authValue
    BYTE*       buffer              // IN/OUT: parameter buffer to be decrypted
)
{
    SESSION* session = SessionGet(handle);  // encrypt session
    // The HMAC key is going to be the concatenation of the session key and any
    // additional key material (like the authValue). The size of both of these
    // is the size of the buffer which can contain a TPMT_HA.
    TPM2B_TYPE(HMAC_KEY,
               (sizeof(extraKey->t.buffer) + sizeof(session->sessionKey.t.buffer)));
    TPM2B_HMAC_KEY key;              // decryption key
    UINT16         cipherSize = 0;  // size of ciphertext

    if(bufferSize < leadingSizeInByte)
    {
        return TPM_RC_INSUFFICIENT;
    }

    // Parameter encryption for a non-2B is not supported.
    if(leadingSizeInByte != 2)
    {
        FAIL_RC(FATAL_ERROR_INTERNAL);
    }

    // Retrieve encrypted data size.
    if(UINT16_Unmarshal(&cipherSize, &buffer, &bufferSize) != TPM_RC_SUCCESS)
    {
        return TPM_RC_INSUFFICIENT;
    }

    if(cipherSize > bufferSize)
    {
        return TPM_RC_SIZE;
    }

    // Compute decryption key by concatenating sessionAuth with extra input key
    MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));
```

```
    MemoryConcat2B(&key.b, &extraKey->b, sizeof(key.t.buffer));

    if(session->symmetric.algorithm == TPM_ALG_XOR)
        // XOR parameter decryption formulation:
        //    XOR(parameter, hash, sessionAuth, nonceNewer, nonceOlder)
        // Call XOR obfuscation function
        CryptXORObfuscation(session->authHashAlg,
                            &key.b,
                            nonceCaller,
                            &(session->nonceTPM.b),
                            (UINT32)cipherSize,
                            buffer);
    else
        // Assume that it is one of the symmetric block ciphers.
        ParmDecryptSym(session->symmetric.algorithm,
                       session->authHashAlg,
                       session->symmetric.keyBits.sym,
                       &key.b,
                       nonceCaller,
                       &session->nonceTPM.b,
                       (UINT32)cipherSize,
                       buffer);

    return TPM_RC_SUCCESS;
}

//*** CryptComputeSymmetricUnique()
// This function computes the unique field in public area for symmetric objects.
void CryptComputeSymmetricUnique(
    TPMT_PUBLIC*    publicArea,  // IN: the object's public area
    TPMT_SENSITIVE* sensitive,   // IN: the associated sensitive area
    TPM2B_DIGEST*   unique       // OUT: unique buffer
)
{
    // For parents (symmetric and derivation), use an HMAC to compute
    // the 'unique' field
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted)
       && IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, decrypt))
    {
        // Unique field is HMAC(sensitive->seedValue, sensitive->sensitive)
        HMAC_STATE hmacState;
        unique->b.size = CryptHmacStart2B(
            &hmacState, publicArea->nameAlg, &sensitive->seedValue.b);
        CryptDigestUpdate2B(&hmacState.hashState, &sensitive->sensitive.any.b);
        CryptHmacEnd2B(&hmacState, &unique->b);
    }
    else
    {
        HASH_STATE hashState;
        // Unique := Hash(sensitive->seedValue || sensitive->sensitive)
        unique->t.size = CryptHashStart(&hashState, publicArea->nameAlg);
        CryptDigestUpdate2B(&hashState, &sensitive->seedValue.b);
        CryptDigestUpdate2B(&hashState, &sensitive->sensitive.any.b);
        CryptHashEnd2B(&hashState, &unique->b);
    }
    return;
}

//*** CryptCreateObject()
// This function creates an object.
// For an asymmetric key, it will create a key pair and, for a parent key, a seed
// value for child protections.
//
// For an symmetric object, (TPM_ALG_SYMCIPHER or TPM_ALG_KEYEDHASH), it will
// create a secret key if the caller did not provide one. It will create a random
// secret seed value that is hashed with the secret value to create the public
```

```
// unique value.
//
// 'publicArea', 'sensitive', and 'sensitiveCreate' are the only required parameters
// and are the only ones that are used by TPM2_Create(). The other parameters
// are optional and are used when the generated Object needs to be deterministic.
// This is the case for both Primary Objects and Derived Objects.
//
// When a seed value is provided, a RAND_STATE will be populated and used for
// all operations in the object generation that require a random number. In the
// simplest case, TPM2_CreatePrimary() will use 'seed', 'label' and 'context' with
// context being the hash of the template. If the Primary Object is in
// the Endorsement hierarchy, it will also populate 'proof' with ehProof.
//
// For derived keys, 'seed' will be the secret value from the parent, 'label' and
// 'context' will be set according to the parameters of TPM2_CreateLoaded() and
// 'hashAlg' will be set which causes the RAND_STATE to be a KDF generator.
//
//  Return Type: TPM_RC
//      TPM_RC_KEY          a provided key is not an allowed value
//      TPM_RC_KEY_SIZE     key size in the public area does not match the size
//                          in the sensitive creation area for a symmetric key
//      TPM_RC_NO_RESULT    unable to get random values (only in derivation)
//      TPM_RC_RANGE        for an RSA key, the exponent is not supported
//      TPM_RC_SIZE         sensitive data size is larger than allowed for the
//                          scheme for a keyed hash object
//      TPM_RC_VALUE        exponent is not prime or could not find a prime using
//                          the provided parameters for an RSA key;
//                          unsupported name algorithm for an ECC key
TPM_RC
CryptCreateObject(OBJECT*                 object,  // IN: new object structure pointer
                  TPMS_SENSITIVE_CREATE* sensitiveCreate,  // IN: sensitive creation
                  RAND_STATE*            rand  // IN: the random number generator
                                               //      to use
)
{
    TPMT_PUBLIC*    publicArea = &object->publicArea;
    TPMT_SENSITIVE* sensitive  = &object->sensitive;
    TPM_RC          result     = TPM_RC_SUCCESS;
    //
    // Set the sensitive type for the object
    sensitive->sensitiveType = publicArea->type;

    // For all objects, copy the initial authorization data
    sensitive->authValue = sensitiveCreate->userAuth;

    // If the TPM is the source of the data, set the size of the provided data to
    // zero so that there's no confusion about what to do.
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sensitiveDataOrigin))
        sensitiveCreate->data.t.size = 0;

    // Generate the key and unique fields for the asymmetric keys and just the
    // sensitive value for symmetric object
    switch(publicArea->type)
    {
#if ALG_RSA
        // Create RSA key
        case TPM_ALG_RSA:
            // RSA uses full object so that it has a place to put the private
            // exponent
            result = CryptRsaGenerateKey(publicArea, sensitive, rand);
            break;
#endif  // ALG_RSA

#if ALG_ECC
        // Create ECC key
        case TPM_ALG_ECC:
```

```
            result = CryptEccGenerateKey(publicArea, sensitive, rand);
            break;
#endif  // ALG_ECC
        case TPM_ALG_SYMCIPHER:
            result = CryptGenerateKeySymmetric(
                publicArea, sensitive, sensitiveCreate, rand);
            break;
        case TPM_ALG_KEYEDHASH:
            result =
                CryptGenerateKeyedHash(publicArea, sensitive, sensitiveCreate, rand);
            break;
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    if(result != TPM_RC_SUCCESS)
        return result;
    // Create the sensitive seed value
    // If this is a primary key in the endorsement hierarchy, stir the DRBG state
    // This implementation uses both shProof and ehProof to make sure that there
    // is no leakage of either.
    if(object->attributes.primary && object->attributes.epsHierarchy)
    {
        DRBG_AdditionalData((DRBG_STATE*)rand, &gp.shProof.b);
        DRBG_AdditionalData((DRBG_STATE*)rand, &gp.ehProof.b);
    }
    // Generate a seedValue that is the size of the digest produced by nameAlg
    sensitive->seedValue.t.size =
        DRBG_Generate(rand,
                      sensitive->seedValue.t.buffer,
                      CryptHashGetDigestSize(publicArea->nameAlg));
    if(g_inFailureMode)
        return TPM_RC_FAILURE;
    else if(sensitive->seedValue.t.size == 0)
        return TPM_RC_NO_RESULT;
    // For symmetric objects, need to compute the unique value for the public area
    if(publicArea->type == TPM_ALG_SYMCIPHER || publicArea->type == TPM_ALG_KEYEDHASH)
    {
        CryptComputeSymmetricUnique(publicArea, sensitive, &publicArea->unique.sym);
    }
    else
    {
        // if this is an asymmetric key and it isn't a parent, then
        // get rid of the seed.
        if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign)
           || !IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted))
            memset(&sensitive->seedValue, 0, sizeof(sensitive->seedValue));
    }
    // Compute the name
    PublicMarshalAndComputeName(publicArea, &object->name);
    return result;
}

//*** CryptGetSignHashAlg()
// Get the hash algorithm of signature from a TPMT_SIGNATURE structure.
// It assumes the signature is not NULL
//  This is a function for easy access
TPMI_ALG_HASH
CryptGetSignHashAlg(TPMT_SIGNATURE* auth  // IN: signature
)
{
    if(auth->sigAlg == TPM_ALG_NULL)
        FAIL(FATAL_ERROR_INTERNAL);

    // Get authHash algorithm based on signing scheme
    switch(auth->sigAlg)
```

```
    {
#if ALG_RSA
        // If RSA is supported, both RSASSA and RSAPSS are required
#  if !defined TPM_ALG_RSASSA || !defined TPM_ALG_RSAPSS
#    error "RSASSA and RSAPSS are required for RSA"
#  endif
        case TPM_ALG_RSASSA:
            return auth->signature.rsassa.hash;
        case TPM_ALG_RSAPSS:
            return auth->signature.rsapss.hash;
#endif  // ALG_RSA

#if ALG_ECC
            // If ECC is defined, ECDSA is mandatory
#  if !ALG_ECDSA
#    error "ECDSA is requried for ECC"
#  endif
        case TPM_ALG_ECDSA:
            // SM2 and ECSCHNORR are optional

#  if ALG_SM2
        case TPM_ALG_SM2:
#  endif
#  if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
#  endif
            //all ECC signatures look the same
            return auth->signature.ecdsa.hash;

#  if ALG_ECDAA
        // Don't know how to verify an ECDAA signature
        case TPM_ALG_ECDAA:
            break;
#  endif

#endif  // ALG_ECC

        case TPM_ALG_HMAC:
            return auth->signature.hmac.hashAlg;

        default:
            break;
    }
    return TPM_ALG_NULL;
}

//*** CryptIsSplitSign()
// This function us used to determine if the signing operation is a split
// signing operation that required a TPM2_Commit().
//
BOOL CryptIsSplitSign(TPM_ALG_ID scheme  // IN: the algorithm selector
)
{
    switch(scheme)
    {
#if ALG_ECDAA
        case TPM_ALG_ECDAA:
            return TRUE;
            break;
#endif  // ALG_ECDAA
        default:
            return FALSE;
            break;
    }
}
```

```
//*** CryptIsAsymSignScheme()
// This function indicates if a scheme algorithm is a sign algorithm.
BOOL CryptIsAsymSignScheme(TPMI_ALG_PUBLIC      publicType,  // IN: Type of the object
                           TPMI_ALG_ASYM_SCHEME scheme       // IN: the scheme
)
{
    BOOL isSignScheme = TRUE;

    switch(publicType)
    {
#if ALG_RSA
        case TPM_ALG_RSA:
            switch(scheme)
            {
#  if !ALG_RSASSA || !ALG_RSAPSS
#    error "RSASSA and PSAPSS required if RSA used."
#  endif
                case TPM_ALG_RSASSA:
                case TPM_ALG_RSAPSS:
                    break;
                default:
                    isSignScheme = FALSE;
                    break;
            }
            break;
#endif  // ALG_RSA

#if ALG_ECC
        // If ECC is implemented ECDSA is required
        case TPM_ALG_ECC:
            switch(scheme)
            {
                // Support for ECDSA is required for ECC
                case TPM_ALG_ECDSA:
#  if ALG_ECDAA  // ECDAA is optional
                case TPM_ALG_ECDAA:
#  endif
#  if ALG_ECSCHNORR  // Schnorr is also optional
                case TPM_ALG_ECSCHNORR:
#  endif
#  if ALG_SM2  // SM2 is optional
                case TPM_ALG_SM2:
#  endif
                    break;
                default:
                    isSignScheme = FALSE;
                    break;
            }
            break;
#endif  // ALG_ECC
        default:
            isSignScheme = FALSE;
            break;
    }
    return isSignScheme;
}

//*** CryptIsAsymDecryptScheme()
// This function indicate if a scheme algorithm is a decrypt algorithm.
BOOL CryptIsAsymDecryptScheme(TPMI_ALG_PUBLIC publicType,  // IN: Type of the object
                              TPMI_ALG_ASYM_SCHEME scheme  // IN: the scheme
)
{
    BOOL isDecryptScheme = TRUE;

    switch(publicType)
```

```
    {
#if ALG_RSA
        case TPM_ALG_RSA:
            switch(scheme)
            {
                case TPM_ALG_RSAES:
                case TPM_ALG_OAEP:
                    break;
                default:
                    isDecryptScheme = FALSE;
                    break;
            }
            break;
#endif  // ALG_RSA

#if ALG_ECC
        // If ECC is implemented ECDH is required
        case TPM_ALG_ECC:
            switch(scheme)
            {
#  if !ALG_ECDH
#    error "ECDH is required for ECC"
#  endif
                case TPM_ALG_ECDH:
#  if ALG_SM2
                case TPM_ALG_SM2:
#  endif
#  if ALG_ECMQV
                case TPM_ALG_ECMQV:
#  endif
                    break;
                default:
                    isDecryptScheme = FALSE;
                    break;
            }
            break;
#endif  // ALG_ECC
        default:
            isDecryptScheme = FALSE;
            break;
    }
    return isDecryptScheme;
}

//*** CryptSelectSignScheme()
// This function is used by the attestation and signing commands.  It implements
// the rules for selecting the signature scheme to use in signing. This function
// requires that the signing key either be TPM_RH_NULL or be loaded.
//
// If a default scheme is defined in object, the default scheme should be chosen,
// otherwise, the input scheme should be chosen.
// In the case that  both object and input scheme has a non-NULL scheme
// algorithm, if the schemes are compatible, the input scheme will be chosen.
//
// This function should not be called if 'signObject->publicArea.type' ==
// ALG_SYMCIPHER.
//
//  Return Type: BOOL
//      TRUE(1)         scheme selected
//      FALSE(0)        both 'scheme' and key's default scheme are empty; or
//                      'scheme' is empty while key's default scheme requires
//                      explicit input scheme (split signing); or
//                      non-empty default key scheme differs from 'scheme'
BOOL CryptSelectSignScheme(OBJECT*          signObject,  // IN: signing key
                           TPMT_SIG_SCHEME* scheme       // IN/OUT: signing scheme
)
```

```
{
    TPMT_SIG_SCHEME* objectScheme;
    TPMT_PUBLIC*     publicArea;
    BOOL             OK;

    // If the signHandle is TPM_RH_NULL, then the NULL scheme is used, regardless
    // of the setting of scheme
    if(signObject == NULL)
    {
        OK                          = TRUE;
        scheme->scheme              = TPM_ALG_NULL;
        scheme->details.any.hashAlg = TPM_ALG_NULL;
    }
    else
    {
        // assignment to save typing.
        publicArea = &signObject->publicArea;

        // A symmetric cipher can be used to encrypt and decrypt but it can't
        // be used for signing
        if(publicArea->type == TPM_ALG_SYMCIPHER)
            return FALSE;
        // Point to the scheme object
        if(CryptIsAsymAlgorithm(publicArea->type))
            objectScheme =
                (TPMT_SIG_SCHEME*)&publicArea->parameters.asymDetail.scheme;
        else
            objectScheme =
                (TPMT_SIG_SCHEME*)&publicArea->parameters.keyedHashDetail.scheme;

        // If the object doesn't have a default scheme, then use the
        // input scheme.
        if(objectScheme->scheme == TPM_ALG_NULL)
        {
            // Input and default can't both be NULL
            OK = (scheme->scheme != TPM_ALG_NULL);
            // Assume that the scheme is compatible with the key. If not,
            // an error will be generated in the signing operation.
        }
        else if(scheme->scheme == TPM_ALG_NULL)
        {
            // input scheme is NULL so use default

            // First, check to see if the default requires that the caller
            // provided scheme data
            OK = !CryptIsSplitSign(objectScheme->scheme);
            if(OK)
            {
                // The object has a scheme and the input is TPM_ALG_NULL so copy
                // the object scheme as the final scheme. It is better to use a
                // structure copy than a copy of the individual fields.
                *scheme = *objectScheme;
            }
        }
        else
        {
            // Both input and object have scheme selectors
            // If the scheme and the hash are not the same then...
            // NOTE: the reason that there is no copy here is that the input
            // might contain extra data for a split signing scheme and that
            // data is not in the object so, it has to be preserved.
            OK =
                (objectScheme->scheme == scheme->scheme)
                && (objectScheme->details.any.hashAlg == scheme->details.any.hashAlg);
        }
    }
```

```
    return OK;
}

//*** CryptSign()
// Sign a digest with asymmetric key or HMAC.
// This function is called by attestation commands and the generic TPM2_Sign
// command.
// This function checks the key scheme and digest size.  It does not
// check if the sign operation is allowed for restricted key.  It should be
// checked before the function is called.
// The function will assert if the key is not a signing key.
//
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       'signScheme' is not compatible with the signing key type
//      TPM_RC_VALUE        'digest' value is greater than the modulus of
//                          'signHandle' or size of 'hashData' does not match hash
//                          algorithm in'signScheme' (for an RSA key);
//                          invalid commit status or failed to generate "r" value
//                          (for an ECC key)
TPM_RC
CryptSign(OBJECT*           signKey,     // IN: signing key
          TPMT_SIG_SCHEME* signScheme,  // IN: sign scheme.
          TPM2B_DIGEST*    digest,      // IN: The digest being signed
          TPMT_SIGNATURE*  signature    // OUT: signature
)
{
    TPM_RC result = TPM_RC_SCHEME;

    // Initialize signature scheme
    signature->sigAlg = signScheme->scheme;

    // If the signature algorithm is TPM_ALG_NULL or the signing key is NULL,
    // then we are done
    if((signature->sigAlg == TPM_ALG_NULL) || (signKey == NULL))
        return TPM_RC_SUCCESS;

    // Initialize signature hash
    // Note: need to do the check for TPM_ALG_NULL first because the null scheme
    // doesn't have a hashAlg member.
    signature->signature.any.hashAlg = signScheme->details.any.hashAlg;

    // perform sign operation based on different key type
    switch(signKey->publicArea.type)
    {
#if ALG_RSA
        case TPM_ALG_RSA:
            result = CryptRsaSign(signature, signKey, digest, NULL);
            break;
#endif  // ALG_RSA
#if ALG_ECC
        case TPM_ALG_ECC:
            // The reason that signScheme is passed to CryptEccSign but not to the
            // other signing methods is that the signing for ECC may be split and
            // need the 'r' value that is in the scheme but not in the signature.
            result = CryptEccSign(
                signature, signKey, digest, (TPMT_ECC_SCHEME*)signScheme, NULL);
            break;
#endif  // ALG_ECC
        case TPM_ALG_KEYEDHASH:
            result = CryptHmacSign(signature, signKey, digest);
            break;
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    return result;
```

```
}

//*** CryptValidateSignature()
// This function is used to verify a signature.  It is called by
// TPM2_VerifySignature() and TPM2_PolicySigned.
//
// Since this operation only requires use of a public key, no consistency
// checks are necessary for the key to signature type because a caller can load
// any public key that they like with any scheme that they like. This routine
// simply makes sure that the signature is correct, whatever the type.
//
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE            the signature is not genuine
//      TPM_RC_SCHEME               the scheme is not supported
//      TPM_RC_HANDLE               an HMAC key was selected but the
//                                  private part of the key is not loaded
TPM_RC
CryptValidateSignature(TPMI_DH_OBJECT  keyHandle,  // IN: The handle of sign key
                       TPM2B_DIGEST*   digest,     // IN: The digest being validated
                       TPMT_SIGNATURE* signature   // IN: signature
)
{
    // NOTE: HandleToObject will either return a pointer to a loaded object or
    // will assert. It will never return a non-valid value. This makes it save
    // to initialize 'publicArea' with the return value from HandleToObject()
    // without checking it first.
    OBJECT*      signObject = HandleToObject(keyHandle);
    TPMT_PUBLIC* publicArea = &signObject->publicArea;
    TPM_RC       result     = TPM_RC_SCHEME;

    // The input unmarshaling should prevent any input signature from being
    // a NULL signature, but just in case
    if(signature->sigAlg == TPM_ALG_NULL)
        return TPM_RC_SIGNATURE;

    switch(publicArea->type)
    {
#if ALG_RSA
        case TPM_ALG_RSA:
        {
            //
            // Call RSA code to verify signature
            result = CryptRsaValidateSignature(signature, signObject, digest);
            break;
        }
#endif  // ALG_RSA

#if ALG_ECC
        case TPM_ALG_ECC:
            result = CryptEccValidateSignature(signature, signObject, digest);
            break;
#endif  // ALG_ECC

        case TPM_ALG_KEYEDHASH:
            if(signObject->attributes.publicOnly)
                result = TPM_RCS_HANDLE;
            else
                result = CryptHMACVerifySignature(signObject, digest, signature);
            break;
        default:
            break;
    }
    return result;
}

//*** CryptGetTestResult
```

```
// This function returns the results of a self-test function.
// Note: the behavior in this function is NOT the correct behavior for a real
// TPM implementation.  An artificial behavior is placed here due to the
// limitation of a software simulation environment.  For the correct behavior,
// consult the part 3 specification for TPM2_GetTestResult().
TPM_RC
CryptGetTestResult(TPM2B_MAX_BUFFER* outData  // OUT: test result data
)
{
    outData->t.size = 0;
    return TPM_RC_SUCCESS;
}

//*** CryptValidateKeys()
// This function is used to verify that the key material of and object is valid.
// For a 'publicOnly' object, the key is verified for size and, if it is an ECC
// key, it is verified to be on the specified curve. For a key with a sensitive
// area, the binding between the public and private parts of the key are verified.
// If the nameAlg of the key is TPM_ALG_NULL, then the size of the sensitive area
// is verified but the public portion is not verified, unless the key is an RSA key.
// For an RSA key, the reason for loading the sensitive area is to use it. The
// only way to use a private RSA key is to compute the private exponent. To compute
// the private exponent, the public modulus is used.
//  Return Type: TPM_RC
//      TPM_RC_BINDING      the public and private parts are not cryptographically
//                          bound
//      TPM_RC_HASH         cannot have a publicOnly key with nameAlg of TPM_ALG_NULL
//      TPM_RC_KEY          the public unique is not valid
//      TPM_RC_KEY_SIZE     the private area key is not valid
//      TPM_RC_TYPE         the types of the sensitive and private parts do not match
TPM_RC
CryptValidateKeys(TPMT_PUBLIC*    publicArea,
                  TPMT_SENSITIVE* sensitive,
                  TPM_RC          blamePublic,
                  TPM_RC          blameSensitive)
{
    TPM_RC              result;
    UINT16              keySizeInBytes;
    UINT16              digestSize = CryptHashGetDigestSize(publicArea->nameAlg);
    TPMU_PUBLIC_PARMS* params     = &publicArea->parameters;
    TPMU_PUBLIC_ID*    unique     = &publicArea->unique;

    if(sensitive != NULL)
    {
        // Make sure that the types of the public and sensitive are compatible
        if(publicArea->type != sensitive->sensitiveType)
            return TPM_RCS_TYPE + blameSensitive;
        // Make sure that the authValue is not bigger than allowed
        // If there is no name algorithm, then the size just needs to be less than
        // the maximum size of the buffer used for authorization. That size check
        // was made during unmarshaling of the sensitive area
        if((sensitive->authValue.t.size) > digestSize && (digestSize > 0))
            return TPM_RCS_SIZE + blameSensitive;
    }
    switch(publicArea->type)
    {
#if ALG_RSA
        case TPM_ALG_RSA:
            keySizeInBytes = BITS_TO_BYTES(params->rsaDetail.keyBits);

            // Regardless of whether there is a sensitive area, the public modulus
            // needs to have the correct size. Otherwise, it can't be used for
            // any public key operation nor can it be used to compute the private
            // exponent.
            // NOTE: This implementation only supports key sizes that are multiples
            // of 1024 bits which means that the MSb of the 0th byte will always be
```

```
            // SET in any prime and in the public modulus.
            if((unique->rsa.t.size != keySizeInBytes)
               || (unique->rsa.t.buffer[0] < 0x80))
                return TPM_RCS_KEY + blamePublic;
            if(params->rsaDetail.exponent != 0 && params->rsaDetail.exponent < 7)
                return TPM_RCS_VALUE + blamePublic;
            if(sensitive != NULL)
            {
                // If there is a sensitive area, it has to be the correct size
                // including having the correct high order bit SET.
                if(((sensitive->sensitive.rsa.t.size * 2) != keySizeInBytes)
                   || (sensitive->sensitive.rsa.t.buffer[0] < 0x80))
                    return TPM_RCS_KEY_SIZE + blameSensitive;
            }
            break;
#endif
#if ALG_ECC
        case TPM_ALG_ECC:
        {
            TPMI_ECC_CURVE curveId;
            curveId        = params->eccDetail.curveID;
            keySizeInBytes = BITS_TO_BYTES(CryptEccGetKeySizeForCurve(curveId));
            if(sensitive == NULL)
            {
                // Validate the public key size
                if(unique->ecc.x.t.size != keySizeInBytes
                   || unique->ecc.y.t.size != keySizeInBytes)
                    return TPM_RCS_KEY + blamePublic;
                if(publicArea->nameAlg != TPM_ALG_NULL)
                {
                    if(!CryptEccIsPointOnCurve(curveId, &unique->ecc))
                        return TPM_RCS_ECC_POINT + blamePublic;
                }
            }
            else
            {
                // If the nameAlg is TPM_ALG_NULL, then only verify that the
                // private part of the key is OK.
                if(!CryptEccIsValidPrivateKey(&sensitive->sensitive.ecc, curveId))
                    return TPM_RCS_KEY_SIZE;
                if(publicArea->nameAlg != TPM_ALG_NULL)
                {
                    // Full key load, verify that the public point belongs to the
                    // private key.
                    TPMS_ECC_POINT toCompare;
                    result = CryptEccPointMultiply(&toCompare,
                                                   curveId,
                                                   NULL,
                                                   &sensitive->sensitive.ecc,
                                                   NULL,
                                                   NULL);
                    if(result != TPM_RC_SUCCESS)
                        return TPM_RCS_BINDING;
                    else
                    {
                        // Make sure that the private key generated the public key.
                        // The input values and the values produced by the point
                        // multiply may not be the same size so adjust the computed
                        // value to match the size of the input value by adding or
                        // removing zeros.
                        AdjustNumberB(&toCompare.x.b, unique->ecc.x.t.size);
                        AdjustNumberB(&toCompare.y.b, unique->ecc.y.t.size);
                        if(!MemoryEqual2B(&unique->ecc.x.b, &toCompare.x.b)
                           || !MemoryEqual2B(&unique->ecc.y.b, &toCompare.y.b))
                            return TPM_RCS_BINDING;
                    }
```

```
                }
            }
            break;
        }
#endif
        default:
            // Checks for SYMCIPHER and KEYEDHASH are largely the same
            // If public area has a nameAlg, then validate the public area size
            // and if there is also a sensitive area, validate the binding

            // For consistency, if the object is public-only just make sure that
            // the unique field is consistent with the name algorithm
            if(sensitive == NULL)
            {
                if(unique->sym.t.size != digestSize)
                    return TPM_RCS_KEY + blamePublic;
            }
            else
            {
                // Make sure that the key size in the sensitive area is consistent.
                if(publicArea->type == TPM_ALG_SYMCIPHER)
                {
                    result = CryptSymKeyValidate(&params->symDetail.sym,
                                                 &sensitive->sensitive.sym);
                    if(result != TPM_RC_SUCCESS)
                        return result + blameSensitive;
                }
                else
                {
                    // For a keyed hash object, the key has to be less than the
                    // smaller of the block size of the hash used in the scheme or
                    // 128 bytes. The worst case value is limited by the
                    // unmarshaling code so the only thing left to be checked is
                    // that it does not exceed the block size of the hash.
                    // by the hash algorithm of the scheme.
                    TPMT_KEYEDHASH_SCHEME* scheme;
                    UINT16                 maxSize;
                    scheme = &params->keyedHashDetail.scheme;
                    if(scheme->scheme == TPM_ALG_XOR)
                    {
                        maxSize = CryptHashGetBlockSize(scheme->details.xor.hashAlg);
                    }
                    else if(scheme->scheme == TPM_ALG_HMAC)
                    {
                        maxSize = CryptHashGetBlockSize(scheme->details.hmac.hashAlg);
                    }
                    else if(scheme->scheme == TPM_ALG_NULL)
                    {
                        // Not signing or xor so must be a data block
                        maxSize = 128;
                    }
                    else
                        return TPM_RCS_SCHEME + blamePublic;
                    if(sensitive->sensitive.bits.t.size > maxSize)
                        return TPM_RCS_KEY_SIZE + blameSensitive;
                }
                // If there is a nameAlg, check the binding
                if(publicArea->nameAlg != TPM_ALG_NULL)
                {
                    TPM2B_DIGEST compare;
                    if(sensitive->seedValue.t.size != digestSize)
                        return TPM_RCS_KEY_SIZE + blameSensitive;

                    CryptComputeSymmetricUnique(publicArea, sensitive, &compare);
                    if(!MemoryEqual2B(&unique->sym.b, &compare.b))
                        return TPM_RC_BINDING;
```

```
                }
            }
            break;
    }
    // For a parent, need to check that the seedValue is the correct size for
    // protections. It should be at least half the size of the nameAlg
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted)
       && IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, decrypt)
       && sensitive != NULL && publicArea->nameAlg != TPM_ALG_NULL)
    {
        if((sensitive->seedValue.t.size < (digestSize / 2))
           || (sensitive->seedValue.t.size > digestSize))
            return TPM_RCS_SIZE + blameSensitive;
    }
    return TPM_RC_SUCCESS;
}

//*** CryptSelectMac()
// This function is used to set the MAC scheme based on the key parameters and
// the input scheme.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       the scheme is not a valid mac scheme
//      TPM_RC_TYPE         the input key is not a type that supports a mac
//      TPM_RC_VALUE        the input scheme and the key scheme are not compatible
TPM_RC
CryptSelectMac(TPMT_PUBLIC* publicArea, TPMI_ALG_MAC_SCHEME* inMac)
{
    TPM_ALG_ID macAlg = TPM_ALG_NULL;
    switch(publicArea->type)
    {
        case TPM_ALG_KEYEDHASH:
        {
            // Local value to keep lines from getting too long
            TPMT_KEYEDHASH_SCHEME* scheme;
            scheme = &publicArea->parameters.keyedHashDetail.scheme;
            // Expect that the scheme is either HMAC or NULL
            if(scheme->scheme != TPM_ALG_NULL)
                macAlg = scheme->details.hmac.hashAlg;
            break;
        }
        case TPM_ALG_SYMCIPHER:
        {
            TPMT_SYM_DEF_OBJECT* scheme;
            scheme = &publicArea->parameters.symDetail.sym;
            // Expect that the scheme is either valid symmetric cipher or NULL
            if(scheme->algorithm != TPM_ALG_NULL)
                macAlg = scheme->mode.sym;
            break;
        }
        default:
            return TPM_RCS_TYPE;
    }
    // If the input value is not TPM_ALG_NULL ...
    if(*inMac != TPM_ALG_NULL)
    {
        // ... then either the scheme in the key must be TPM_ALG_NULL or the input
        // value must match
        if((macAlg != TPM_ALG_NULL) && (*inMac != macAlg))
            return TPM_RCS_VALUE;
    }
    else
    {
        // Since the input value is TPM_ALG_NULL, then the key value can't be
        // TPM_ALG_NULL
        if(macAlg == TPM_ALG_NULL)
            return TPM_RCS_VALUE;
```

```
        *inMac = macAlg;
    }
    if(!CryptMacIsValidForKey(publicArea->type, *inMac, FALSE))
        return TPM_RCS_SCHEME;
    return TPM_RC_SUCCESS;
}

//*** CryptMacIsValidForKey()
// Check to see if the key type is compatible with the mac type
BOOL CryptMacIsValidForKey(TPM_ALG_ID keyType, TPM_ALG_ID macAlg, BOOL flag)
{
    switch(keyType)
    {
        case TPM_ALG_KEYEDHASH:
            return CryptHashIsValidAlg(macAlg, flag);
            break;
        case TPM_ALG_SYMCIPHER:
            return CryptSmacIsValidAlg(macAlg, flag);
            break;
        default:
            break;
    }
    return FALSE;
}

//*** CryptSmacIsValidAlg()
// This function is used to test if an algorithm is a supported SMAC algorithm. It
// needs to be updated as new algorithms are added.
BOOL CryptSmacIsValidAlg(TPM_ALG_ID alg,
                         BOOL       FLAG  // IN: Indicates if TPM_ALG_NULL is valid
)
{
    switch(alg)
    {
#if ALG_CMAC
        case TPM_ALG_CMAC:
            return TRUE;
            break;
#endif
        case TPM_ALG_NULL:
            return FLAG;
            break;
        default:
            return FALSE;
    }
}

//*** CryptSymModeIsValid()
// Function checks to see if an algorithm ID is a valid, symmetric block cipher
// mode for the TPM. If 'flag' is SET, them TPM_ALG_NULL is a valid mode.
// not include the modes used for SMAC
BOOL CryptSymModeIsValid(TPM_ALG_ID mode, BOOL flag)
{
    switch(mode)
    {
#if ALG_CTR
        case TPM_ALG_CTR:
#endif  // ALG_CTR
#if ALG_OFB
        case TPM_ALG_OFB:
#endif  // ALG_OFB
#if ALG_CBC
        case TPM_ALG_CBC:
#endif  // ALG_CBC
#if ALG_CFB
        case TPM_ALG_CFB:
```

```
#endif  // ALG_CFB
#if ALG_ECB
        case TPM_ALG_ECB:
#endif  // ALG_ECB
            return TRUE;
        case TPM_ALG_NULL:
            return flag;
            break;
        default:
            break;
    }
    return FALSE;
}
```

7.152 /tpm/src/crypt/PrimeData.c

```
#include "Tpm.h"

// This table is the product of all of the primes up to 1000.
// Checking to see if there is a GCD between a prime candidate
// and this number will eliminate many prime candidates from
// consideration before running Miller-Rabin on the result.

const CRYPT_INT_BUF(smallprimecomp, 43 * RADIX_BITS) s_CompositeOfSmallPrimes_ =
    {44, 44, {0x2ED42696, 0x2BBFA177, 0x4820594F, 0xF73F4841, 0xBFAC313A, 0xCAC3EB81,
              0xF6F26BF8, 0x7FAB5061, 0x59746FB7, 0xF71377F6, 0x3B19855B, 0xCBD03132,
              0xBB92EF1B, 0x3AC3152C, 0xE87C8273, 0xC0AE0E69, 0x74A9E295, 0x448CCE86,
              0x63CA1907, 0x8A0BF944, 0xF8CC3BE0, 0xC26F0AF5, 0xC501C02F, 0x6579441A,
              0xD1099CDA, 0x6BC76A00, 0xC81A3228, 0xBFB1AB25, 0x70FA3841, 0x51B3D076,
              0xCC2359ED, 0xD9EE0769, 0x75E47AF0, 0xD45FF31E, 0x52CCE4F6, 0x04DBC891,
              0x96658ED2, 0x1753EFE5, 0x3AE4A5A6, 0x8FD4A97F, 0x8B15E7EB, 0x0243C3E1,
              0xE0F0C31D, 0x0000000B}};

const Crypt_Int* s_CompositeOfSmallPrimes =
    (const Crypt_Int*)&s_CompositeOfSmallPrimes_;

// This table contains a bit for each of the odd values between 1 and 2^16 + 1.
// This table allows fast checking of the primes in that range.
// Don't change the size of this table unless you are prepared to do redo
// IsPrimeInt().

const uint32_t      s_LastPrimeInTable = 65537;
const uint32_t      s_PrimeTableSize   = 4097;
const uint32_t      s_PrimesInTable    = 6542;
const unsigned char s_PrimeTable[] =
    {0x6e, 0xcb, 0xb4, 0x64, 0x9a, 0x12, 0x6d, 0x81, 0x32, 0x4c, 0x4a, 0x86, 0x0d,
     0x82, 0x96, 0x21, 0xc9, 0x34, 0x04, 0x5a, 0x20, 0x61, 0x89, 0xa4, 0x44, 0x11,
     0x86, 0x29, 0xd1, 0x82, 0x28, 0x4a, 0x30, 0x40, 0x42, 0x32, 0x21, 0x99, 0x34,
     0x08, 0x4b, 0x06, 0x25, 0x42, 0x84, 0x48, 0x8a, 0x14, 0x05, 0x42, 0x30, 0x6c,
     0x08, 0xb4, 0x40, 0x0b, 0xa0, 0x08, 0x51, 0x12, 0x28, 0x89, 0x04, 0x65, 0x98,
     0x30, 0x4c, 0x80, 0x96, 0x44, 0x12, 0x80, 0x21, 0x42, 0x12, 0x41, 0xc9, 0x04,
     0x21, 0xc0, 0x32, 0x2d, 0x98, 0x00, 0x00, 0x49, 0x04, 0x08, 0x81, 0x96, 0x68,
     0x82, 0xb0, 0x25, 0x08, 0x22, 0x48, 0x89, 0xa2, 0x40, 0x59, 0x26, 0x04, 0x90,
     0x06, 0x40, 0x43, 0x30, 0x44, 0x92, 0x00, 0x69, 0x10, 0x82, 0x08, 0x08, 0xa4,
     0x0d, 0x41, 0x12, 0x60, 0xc0, 0x00, 0x24, 0xd2, 0x22, 0x61, 0x08, 0x84, 0x04,
     0x1b, 0x82, 0x01, 0xd3, 0x10, 0x01, 0x02, 0xa0, 0x44, 0xc0, 0x22, 0x60, 0x91,
     0x14, 0x0c, 0x40, 0xa6, 0x04, 0xd2, 0x94, 0x20, 0x09, 0x94, 0x20, 0x52, 0x00,
     0x08, 0x10, 0xa2, 0x4c, 0x00, 0x82, 0x01, 0x51, 0x10, 0x08, 0x8b, 0xa4, 0x25,
     0x9a, 0x30, 0x44, 0x81, 0x10, 0x4c, 0x03, 0x02, 0x25, 0x52, 0x80, 0x08, 0x49,
     0x84, 0x20, 0x50, 0x32, 0x00, 0x18, 0xa2, 0x40, 0x11, 0x24, 0x28, 0x01, 0x84,
     0x01, 0x01, 0xa0, 0x41, 0x0a, 0x12, 0x45, 0x00, 0x36, 0x08, 0x00, 0x26, 0x29,
     0x83, 0x82, 0x61, 0xc0, 0x80, 0x04, 0x10, 0x10, 0x6d, 0x00, 0x22, 0x48, 0x58,
     0x26, 0x0c, 0xc2, 0x10, 0x48, 0x89, 0x24, 0x20, 0x58, 0x20, 0x45, 0x88, 0x24,
     0x00, 0x19, 0x02, 0x25, 0xc0, 0x10, 0x68, 0x08, 0x14, 0x01, 0xca, 0x32, 0x28,
     0x80, 0x00, 0x04, 0x4b, 0x26, 0x00, 0x13, 0x90, 0x60, 0x82, 0x80, 0x25, 0xd0,
```

```
0x00, 0x01, 0x10, 0x32, 0x0c, 0x43, 0x86, 0x21, 0x11, 0x00, 0x08, 0x43, 0x24,
0x04, 0x48, 0x10, 0x0c, 0x90, 0x92, 0x00, 0x43, 0x20, 0x2d, 0x00, 0x06, 0x09,
0x88, 0x24, 0x40, 0xc0, 0x32, 0x09, 0x09, 0x82, 0x00, 0x53, 0x80, 0x08, 0x80,
0x96, 0x41, 0x81, 0x00, 0x40, 0x48, 0x10, 0x48, 0x08, 0x96, 0x48, 0x58, 0x20,
0x29, 0xc3, 0x80, 0x20, 0x02, 0x94, 0x60, 0x92, 0x00, 0x20, 0x81, 0x22, 0x44,
0x10, 0xa0, 0x05, 0x40, 0x90, 0x01, 0x49, 0x20, 0x04, 0x0a, 0x00, 0x24, 0x89,
0x34, 0x48, 0x13, 0x80, 0x2c, 0xc0, 0x82, 0x29, 0x00, 0x24, 0x45, 0x08, 0x00,
0x08, 0x98, 0x36, 0x04, 0x52, 0x84, 0x04, 0xd0, 0x04, 0x00, 0x8a, 0x90, 0x44,
0x82, 0x32, 0x65, 0x18, 0x90, 0x00, 0x0a, 0x02, 0x01, 0x40, 0x02, 0x28, 0x40,
0xa4, 0x04, 0x92, 0x30, 0x04, 0x11, 0x86, 0x08, 0x42, 0x00, 0x2c, 0x52, 0x04,
0x08, 0xc9, 0x84, 0x60, 0x48, 0x12, 0x09, 0x99, 0x24, 0x44, 0x00, 0x24, 0x00,
0x03, 0x14, 0x21, 0x00, 0x10, 0x01, 0x1a, 0x32, 0x05, 0x88, 0x20, 0x40, 0x40,
0x06, 0x09, 0xc3, 0x84, 0x40, 0x01, 0x30, 0x60, 0x18, 0x02, 0x68, 0x11, 0x90,
0x0c, 0x02, 0xa2, 0x04, 0x00, 0x86, 0x29, 0x89, 0x14, 0x24, 0x82, 0x02, 0x41,
0x08, 0x80, 0x04, 0x19, 0x80, 0x08, 0x10, 0x12, 0x68, 0x42, 0xa4, 0x04, 0x00,
0x02, 0x61, 0x10, 0x06, 0x0c, 0x10, 0x00, 0x01, 0x12, 0x10, 0x20, 0x03, 0x94,
0x21, 0x42, 0x12, 0x65, 0x18, 0x94, 0x0c, 0x0a, 0x04, 0x28, 0x01, 0x14, 0x29,
0x0a, 0xa4, 0x40, 0xd0, 0x00, 0x40, 0x01, 0x90, 0x04, 0x41, 0x20, 0x2d, 0x40,
0x82, 0x48, 0xc1, 0x20, 0x00, 0x10, 0x30, 0x01, 0x08, 0x24, 0x04, 0x59, 0x84,
0x24, 0x00, 0x02, 0x29, 0x82, 0x00, 0x61, 0x58, 0x02, 0x48, 0x81, 0x16, 0x48,
0x10, 0x00, 0x21, 0x11, 0x06, 0x00, 0xca, 0xa0, 0x40, 0x02, 0x00, 0x04, 0x91,
0xb0, 0x00, 0x42, 0x04, 0x0c, 0x81, 0x06, 0x09, 0x48, 0x14, 0x25, 0x92, 0x20,
0x25, 0x11, 0xa0, 0x00, 0x0a, 0x86, 0x0c, 0xc1, 0x02, 0x48, 0x00, 0x20, 0x45,
0x08, 0x32, 0x00, 0x98, 0x06, 0x04, 0x13, 0x22, 0x00, 0x82, 0x04, 0x48, 0x81,
0x14, 0x44, 0x82, 0x12, 0x24, 0x18, 0x10, 0x40, 0x43, 0x80, 0x28, 0xd0, 0x04,
0x20, 0x81, 0x24, 0x64, 0xd8, 0x00, 0x2c, 0x09, 0x12, 0x08, 0x41, 0xa2, 0x00,
0x00, 0x02, 0x41, 0xca, 0x20, 0x41, 0xc0, 0x10, 0x01, 0x18, 0xa4, 0x04, 0x18,
0xa4, 0x20, 0x12, 0x94, 0x20, 0x83, 0xa0, 0x40, 0x02, 0x32, 0x44, 0x80, 0x04,
0x00, 0x18, 0x00, 0x0c, 0x40, 0x86, 0x60, 0x8a, 0x00, 0x64, 0x88, 0x12, 0x05,
0x01, 0x82, 0x00, 0x4a, 0xa2, 0x01, 0xc1, 0x10, 0x61, 0x09, 0x04, 0x01, 0x88,
0x00, 0x60, 0x01, 0xb4, 0x40, 0x08, 0x06, 0x01, 0x03, 0x80, 0x08, 0x40, 0x94,
0x04, 0x8a, 0x20, 0x29, 0x80, 0x02, 0x0c, 0x52, 0x02, 0x01, 0x42, 0x84, 0x00,
0x80, 0x84, 0x64, 0x02, 0x32, 0x48, 0x00, 0x30, 0x44, 0x40, 0x22, 0x21, 0x00,
0x02, 0x08, 0xc3, 0xa0, 0x04, 0xd0, 0x20, 0x40, 0x18, 0x16, 0x40, 0x40, 0x00,
0x28, 0x52, 0x90, 0x08, 0x82, 0x14, 0x01, 0x18, 0x10, 0x08, 0x09, 0x82, 0x40,
0x0a, 0xa0, 0x20, 0x93, 0x80, 0x08, 0xc0, 0x00, 0x20, 0x52, 0x00, 0x05, 0x01,
0x10, 0x40, 0x11, 0x06, 0x0c, 0x82, 0x00, 0x00, 0x4b, 0x90, 0x44, 0x9a, 0x00,
0x28, 0x80, 0x90, 0x04, 0x4a, 0x06, 0x09, 0x43, 0x02, 0x28, 0x00, 0x34, 0x01,
0x18, 0x00, 0x65, 0x09, 0x80, 0x44, 0x03, 0x00, 0x24, 0x02, 0x82, 0x61, 0x48,
0x14, 0x41, 0x00, 0x12, 0x28, 0x00, 0x34, 0x08, 0x51, 0x04, 0x05, 0x12, 0x90,
0x28, 0x89, 0x84, 0x60, 0x12, 0x10, 0x49, 0x10, 0x26, 0x40, 0x49, 0x82, 0x00,
0x91, 0x10, 0x01, 0x0a, 0x24, 0x40, 0x88, 0x10, 0x4c, 0x10, 0x04, 0x00, 0x50,
0xa2, 0x2c, 0x40, 0x90, 0x48, 0x0a, 0xb0, 0x01, 0x50, 0x12, 0x08, 0x00, 0xa4,
0x04, 0x09, 0xa0, 0x28, 0x92, 0x02, 0x00, 0x43, 0x10, 0x21, 0x02, 0x20, 0x41,
0x81, 0x32, 0x00, 0x08, 0x04, 0x0c, 0x52, 0x00, 0x21, 0x49, 0x84, 0x20, 0x10,
0x02, 0x01, 0x81, 0x10, 0x48, 0x40, 0x22, 0x01, 0x01, 0x84, 0x69, 0xc1, 0x30,
0x01, 0xc8, 0x02, 0x44, 0x88, 0x00, 0x0c, 0x01, 0x02, 0x2d, 0xc0, 0x12, 0x61,
0x00, 0xa0, 0x00, 0xc0, 0x30, 0x40, 0x01, 0x12, 0x08, 0x0b, 0x20, 0x00, 0x80,
0x94, 0x40, 0x01, 0x84, 0x40, 0x00, 0x32, 0x00, 0x10, 0x84, 0x00, 0x0b, 0x24,
0x00, 0x01, 0x06, 0x29, 0x8a, 0x84, 0x41, 0x80, 0x10, 0x08, 0x08, 0x94, 0x4c,
0x03, 0x80, 0x01, 0x40, 0x96, 0x40, 0x41, 0x20, 0x20, 0x50, 0x22, 0x25, 0x89,
0xa2, 0x40, 0x40, 0xa4, 0x20, 0x02, 0x86, 0x28, 0x01, 0x20, 0x21, 0x4a, 0x10,
0x08, 0x00, 0x14, 0x08, 0x40, 0x04, 0x25, 0x42, 0x02, 0x21, 0x43, 0x10, 0x04,
0x92, 0x00, 0x21, 0x11, 0xa0, 0x4c, 0x18, 0x22, 0x09, 0x03, 0x84, 0x41, 0x89,
0x10, 0x04, 0x82, 0x22, 0x24, 0x01, 0x14, 0x08, 0x08, 0x84, 0x08, 0xc1, 0x00,
0x09, 0x42, 0xb0, 0x41, 0x8a, 0x02, 0x00, 0x80, 0x36, 0x04, 0x49, 0xa0, 0x24,
0x91, 0x00, 0x00, 0x02, 0x94, 0x41, 0x92, 0x02, 0x01, 0x08, 0x06, 0x08, 0x09,
0x00, 0x01, 0xd0, 0x16, 0x28, 0x89, 0x80, 0x60, 0x00, 0x00, 0x68, 0x01, 0x90,
0x0c, 0x50, 0x20, 0x01, 0x40, 0x80, 0x40, 0x42, 0x30, 0x41, 0x00, 0x20, 0x25,
0x81, 0x06, 0x40, 0x49, 0x00, 0x08, 0x01, 0x12, 0x49, 0x00, 0xa0, 0x20, 0x18,
0x30, 0x05, 0x01, 0xa6, 0x00, 0x10, 0x24, 0x28, 0x00, 0x02, 0x20, 0xc8, 0x20,
0x00, 0x88, 0x12, 0x0c, 0x90, 0x92, 0x00, 0x02, 0x26, 0x01, 0x42, 0x16, 0x49,
0x00, 0x04, 0x24, 0x42, 0x02, 0x01, 0x88, 0x80, 0x0c, 0x1a, 0x80, 0x08, 0x10,
0x00, 0x60, 0x02, 0x94, 0x44, 0x88, 0x00, 0x69, 0x11, 0x30, 0x08, 0x12, 0xa0,
0x24, 0x13, 0x84, 0x00, 0x82, 0x00, 0x65, 0xc0, 0x10, 0x28, 0x00, 0x30, 0x04,
0x03, 0x20, 0x01, 0x11, 0x06, 0x01, 0xc8, 0x80, 0x00, 0xc2, 0x20, 0x08, 0x10,
```

```
0x82, 0x0c, 0x13, 0x02, 0x0c, 0x52, 0x06, 0x40, 0x00, 0xb0, 0x61, 0x40, 0x10,
0x01, 0x98, 0x86, 0x04, 0x10, 0x84, 0x08, 0x92, 0x14, 0x60, 0x41, 0x80, 0x41,
0x1a, 0x10, 0x04, 0x81, 0x22, 0x40, 0x41, 0x20, 0x29, 0x52, 0x00, 0x41, 0x08,
0x34, 0x60, 0x10, 0x00, 0x28, 0x01, 0x10, 0x40, 0x00, 0x84, 0x08, 0x42, 0x90,
0x20, 0x48, 0x04, 0x04, 0x52, 0x02, 0x00, 0x08, 0x20, 0x04, 0x00, 0x82, 0x0d,
0x00, 0x82, 0x40, 0x02, 0x10, 0x05, 0x48, 0x20, 0x40, 0x99, 0x00, 0x00, 0x01,
0x06, 0x24, 0xc0, 0x00, 0x68, 0x82, 0x04, 0x21, 0x12, 0x10, 0x44, 0x08, 0x04,
0x00, 0x40, 0xa6, 0x20, 0xd0, 0x16, 0x09, 0xc9, 0x24, 0x41, 0x02, 0x20, 0x0c,
0x09, 0x92, 0x40, 0x12, 0x00, 0x00, 0x40, 0x00, 0x09, 0x43, 0x84, 0x20, 0x98,
0x02, 0x01, 0x11, 0x24, 0x00, 0x43, 0x24, 0x00, 0x03, 0x90, 0x08, 0x41, 0x30,
0x24, 0x58, 0x20, 0x4c, 0x80, 0x82, 0x08, 0x10, 0x24, 0x25, 0x81, 0x06, 0x41,
0x09, 0x10, 0x20, 0x18, 0x10, 0x44, 0x80, 0x10, 0x00, 0x4a, 0x24, 0x0d, 0x01,
0x94, 0x28, 0x80, 0x30, 0x00, 0xc0, 0x02, 0x60, 0x10, 0x84, 0x0c, 0x02, 0x00,
0x09, 0x02, 0x82, 0x01, 0x08, 0x10, 0x04, 0xc2, 0x20, 0x68, 0x09, 0x06, 0x04,
0x18, 0x00, 0x00, 0x11, 0x90, 0x08, 0x0b, 0x10, 0x21, 0x82, 0x02, 0x0c, 0x10,
0xb6, 0x08, 0x00, 0x26, 0x00, 0x41, 0x02, 0x01, 0x4a, 0x24, 0x21, 0x1a, 0x20,
0x24, 0x80, 0x00, 0x44, 0x02, 0x00, 0x2d, 0x40, 0x02, 0x00, 0x8b, 0x94, 0x20,
0x10, 0x00, 0x20, 0x90, 0xa6, 0x40, 0x13, 0x00, 0x2c, 0x11, 0x86, 0x61, 0x01,
0x80, 0x41, 0x10, 0x02, 0x04, 0x81, 0x30, 0x48, 0x48, 0x20, 0x28, 0x50, 0x80,
0x21, 0x8a, 0x10, 0x04, 0x08, 0x10, 0x09, 0x10, 0x10, 0x48, 0x42, 0xa0, 0x0c,
0x82, 0x92, 0x60, 0xc0, 0x20, 0x05, 0xd2, 0x20, 0x40, 0x01, 0x00, 0x04, 0x08,
0x82, 0x2d, 0x82, 0x02, 0x00, 0x48, 0x80, 0x41, 0x48, 0x10, 0x00, 0x91, 0x04,
0x04, 0x03, 0x84, 0x00, 0xc2, 0x04, 0x68, 0x00, 0x00, 0x64, 0xc0, 0x22, 0x40,
0x08, 0x32, 0x44, 0x09, 0x86, 0x00, 0x91, 0x02, 0x28, 0x01, 0x00, 0x64, 0x48,
0x00, 0x24, 0x10, 0x90, 0x00, 0x43, 0x00, 0x21, 0x52, 0x86, 0x41, 0x8b, 0x90,
0x20, 0x40, 0x20, 0x08, 0x88, 0x04, 0x44, 0x13, 0x20, 0x00, 0x02, 0x84, 0x60,
0x81, 0x90, 0x24, 0x40, 0x30, 0x00, 0x08, 0x10, 0x08, 0x08, 0x02, 0x01, 0x10,
0x04, 0x20, 0x43, 0xb4, 0x40, 0x90, 0x12, 0x68, 0x01, 0x80, 0x4c, 0x18, 0x00,
0x08, 0xc0, 0x12, 0x49, 0x40, 0x10, 0x24, 0x1a, 0x00, 0x41, 0x89, 0x24, 0x4c,
0x10, 0x00, 0x04, 0x52, 0x10, 0x09, 0x4a, 0x20, 0x41, 0x48, 0x22, 0x69, 0x11,
0x14, 0x08, 0x10, 0x06, 0x24, 0x80, 0x84, 0x28, 0x00, 0x10, 0x00, 0x40, 0x10,
0x01, 0x08, 0x26, 0x08, 0x48, 0x06, 0x28, 0x00, 0x14, 0x01, 0x42, 0x84, 0x04,
0x0a, 0x20, 0x00, 0x01, 0x82, 0x08, 0x00, 0x82, 0x24, 0x12, 0x04, 0x40, 0x40,
0xa0, 0x40, 0x90, 0x10, 0x04, 0x90, 0x22, 0x40, 0x10, 0x20, 0x2c, 0x80, 0x10,
0x28, 0x43, 0x00, 0x04, 0x58, 0x00, 0x01, 0x81, 0x10, 0x48, 0x09, 0x20, 0x21,
0x83, 0x04, 0x00, 0x42, 0xa4, 0x44, 0x00, 0x00, 0x6c, 0x10, 0xa0, 0x44, 0x48,
0x80, 0x00, 0x83, 0x80, 0x48, 0xc9, 0x00, 0x00, 0x00, 0x02, 0x05, 0x10, 0xb0,
0x04, 0x13, 0x04, 0x29, 0x10, 0x92, 0x40, 0x08, 0x04, 0x44, 0x82, 0x22, 0x00,
0x19, 0x20, 0x00, 0x19, 0x20, 0x01, 0x81, 0x90, 0x60, 0x8a, 0x00, 0x41, 0xc0,
0x02, 0x45, 0x10, 0x04, 0x00, 0x02, 0xa2, 0x09, 0x40, 0x10, 0x21, 0x49, 0x20,
0x01, 0x42, 0x30, 0x2c, 0x00, 0x14, 0x44, 0x01, 0x22, 0x04, 0x02, 0x92, 0x08,
0x89, 0x04, 0x21, 0x80, 0x10, 0x05, 0x01, 0x20, 0x40, 0x41, 0x80, 0x04, 0x00,
0x12, 0x09, 0x40, 0xb0, 0x64, 0x58, 0x32, 0x01, 0x08, 0x90, 0x00, 0x41, 0x04,
0x09, 0xc1, 0x80, 0x61, 0x08, 0x90, 0x00, 0x9a, 0x00, 0x24, 0x01, 0x12, 0x08,
0x02, 0x26, 0x05, 0x82, 0x06, 0x08, 0x08, 0x00, 0x20, 0x48, 0x20, 0x00, 0x18,
0x24, 0x48, 0x03, 0x02, 0x00, 0x11, 0x00, 0x09, 0x00, 0x84, 0x01, 0x4a, 0x10,
0x01, 0x98, 0x00, 0x04, 0x18, 0x86, 0x00, 0xc0, 0x00, 0x20, 0x81, 0x80, 0x04,
0x10, 0x30, 0x05, 0x00, 0xb4, 0x0c, 0x4a, 0x82, 0x29, 0x91, 0x02, 0x28, 0x00,
0x20, 0x44, 0xc0, 0x00, 0x2c, 0x91, 0x80, 0x40, 0x01, 0xa2, 0x00, 0x12, 0x04,
0x09, 0xc3, 0x20, 0x00, 0x08, 0x02, 0x0c, 0x10, 0x22, 0x04, 0x00, 0x00, 0x2c,
0x11, 0x86, 0x00, 0xc0, 0x00, 0x00, 0x12, 0x32, 0x40, 0x89, 0x80, 0x40, 0x40,
0x02, 0x05, 0x50, 0x86, 0x60, 0x82, 0xa4, 0x60, 0x0a, 0x12, 0x4d, 0x80, 0x90,
0x08, 0x12, 0x80, 0x09, 0x02, 0x14, 0x48, 0x01, 0x24, 0x20, 0x8a, 0x00, 0x44,
0x90, 0x04, 0x04, 0x01, 0x02, 0x00, 0xd1, 0x12, 0x00, 0x0a, 0x04, 0x40, 0x00,
0x32, 0x21, 0x81, 0x24, 0x08, 0x19, 0x84, 0x20, 0x02, 0x04, 0x08, 0x89, 0x80,
0x24, 0x02, 0x02, 0x68, 0x18, 0x82, 0x44, 0x42, 0x00, 0x21, 0x40, 0x00, 0x28,
0x01, 0x80, 0x45, 0x82, 0x20, 0x40, 0x11, 0x80, 0x0c, 0x02, 0x00, 0x24, 0x40,
0x90, 0x01, 0x40, 0x20, 0x20, 0x50, 0x20, 0x28, 0x19, 0x00, 0x40, 0x09, 0x20,
0x08, 0x80, 0x04, 0x60, 0x40, 0x80, 0x20, 0x08, 0x30, 0x49, 0x09, 0x34, 0x00,
0x11, 0x24, 0x24, 0x82, 0x00, 0x41, 0xc2, 0x00, 0x04, 0x92, 0x02, 0x24, 0x80,
0x00, 0x0c, 0x02, 0xa0, 0x00, 0x01, 0x06, 0x60, 0x41, 0x04, 0x21, 0xd0, 0x00,
0x01, 0x01, 0x00, 0x48, 0x12, 0x84, 0x04, 0x91, 0x12, 0x08, 0x00, 0x24, 0x44,
0x00, 0x12, 0x41, 0x18, 0x26, 0x0c, 0x41, 0x80, 0x00, 0x52, 0x04, 0x20, 0x09,
0x00, 0x24, 0x90, 0x20, 0x48, 0x18, 0x02, 0x00, 0x03, 0xa2, 0x09, 0xd0, 0x14,
0x00, 0x8a, 0x84, 0x25, 0x4a, 0x00, 0x20, 0x98, 0x14, 0x40, 0x00, 0xa2, 0x05,
0x00, 0x00, 0x00, 0x40, 0x14, 0x01, 0x58, 0x20, 0x2c, 0x80, 0x84, 0x00, 0x09,
```

```
0x20, 0x20, 0x91, 0x02, 0x08, 0x02, 0xb0, 0x41, 0x08, 0x30, 0x00, 0x09, 0x10,
0x00, 0x18, 0x02, 0x21, 0x02, 0x02, 0x00, 0x00, 0x24, 0x44, 0x08, 0x12, 0x60,
0x00, 0xb2, 0x44, 0x12, 0x02, 0x0c, 0xc0, 0x80, 0x40, 0xc8, 0x20, 0x04, 0x50,
0x20, 0x05, 0x00, 0xb0, 0x04, 0x0b, 0x04, 0x29, 0x53, 0x00, 0x61, 0x48, 0x30,
0x00, 0x82, 0x20, 0x29, 0x00, 0x16, 0x00, 0x53, 0x22, 0x20, 0x43, 0x10, 0x48,
0x00, 0x80, 0x04, 0xd2, 0x00, 0x40, 0x00, 0xa2, 0x44, 0x03, 0x80, 0x29, 0x00,
0x04, 0x08, 0xc0, 0x04, 0x64, 0x40, 0x30, 0x28, 0x09, 0x84, 0x44, 0x50, 0x80,
0x21, 0x02, 0x92, 0x00, 0xc0, 0x10, 0x60, 0x88, 0x22, 0x08, 0x80, 0x00, 0x00,
0x18, 0x84, 0x04, 0x83, 0x96, 0x00, 0x81, 0x20, 0x05, 0x02, 0x00, 0x45, 0x88,
0x84, 0x00, 0x51, 0x20, 0x20, 0x51, 0x86, 0x41, 0x4b, 0x94, 0x00, 0x80, 0x00,
0x08, 0x11, 0x20, 0x4c, 0x58, 0x80, 0x04, 0x03, 0x06, 0x20, 0x89, 0x00, 0x05,
0x08, 0x22, 0x05, 0x90, 0x00, 0x40, 0x00, 0x82, 0x09, 0x50, 0x00, 0x00, 0x00,
0xa0, 0x41, 0xc2, 0x20, 0x08, 0x00, 0x16, 0x08, 0x40, 0x26, 0x21, 0xd0, 0x90,
0x08, 0x81, 0x90, 0x41, 0x00, 0x02, 0x44, 0x08, 0x10, 0x0c, 0x0a, 0x86, 0x09,
0x90, 0x04, 0x00, 0xc8, 0xa0, 0x04, 0x08, 0x30, 0x20, 0x89, 0x84, 0x00, 0x11,
0x22, 0x2c, 0x40, 0x00, 0x08, 0x02, 0xb0, 0x01, 0x48, 0x02, 0x01, 0x09, 0x20,
0x04, 0x03, 0x04, 0x00, 0x80, 0x02, 0x60, 0x42, 0x30, 0x21, 0x4a, 0x10, 0x44,
0x09, 0x02, 0x00, 0x01, 0x24, 0x00, 0x12, 0x82, 0x21, 0x80, 0xa4, 0x20, 0x10,
0x02, 0x04, 0x91, 0xa0, 0x40, 0x18, 0x04, 0x00, 0x02, 0x06, 0x69, 0x09, 0x00,
0x05, 0x58, 0x02, 0x01, 0x00, 0x00, 0x48, 0x00, 0x00, 0x00, 0x03, 0x92, 0x20,
0x00, 0x34, 0x01, 0xc8, 0x20, 0x48, 0x08, 0x30, 0x08, 0x42, 0x80, 0x20, 0x91,
0x90, 0x68, 0x01, 0x04, 0x40, 0x12, 0x02, 0x61, 0x00, 0x12, 0x08, 0x01, 0xa0,
0x00, 0x11, 0x04, 0x21, 0x48, 0x04, 0x24, 0x92, 0x00, 0x0c, 0x01, 0x84, 0x04,
0x00, 0x00, 0x01, 0x12, 0x96, 0x40, 0x01, 0xa0, 0x41, 0x88, 0x22, 0x28, 0x88,
0x00, 0x44, 0x42, 0x80, 0x24, 0x12, 0x14, 0x01, 0x42, 0x90, 0x60, 0x1a, 0x10,
0x04, 0x81, 0x10, 0x48, 0x08, 0x06, 0x29, 0x83, 0x02, 0x40, 0x02, 0x24, 0x64,
0x80, 0x10, 0x05, 0x80, 0x10, 0x40, 0x02, 0x02, 0x08, 0x42, 0x84, 0x01, 0x09,
0x20, 0x04, 0x50, 0x00, 0x60, 0x11, 0x30, 0x40, 0x13, 0x02, 0x04, 0x81, 0x00,
0x09, 0x08, 0x20, 0x45, 0x4a, 0x10, 0x61, 0x90, 0x26, 0x0c, 0x08, 0x02, 0x21,
0x91, 0x00, 0x60, 0x02, 0x04, 0x00, 0x02, 0x00, 0x0c, 0x08, 0x06, 0x08, 0x48,
0x84, 0x08, 0x11, 0x02, 0x00, 0x80, 0xa4, 0x00, 0x5a, 0x20, 0x00, 0x88, 0x04,
0x04, 0x02, 0x00, 0x09, 0x00, 0x14, 0x08, 0x49, 0x14, 0x20, 0xc8, 0x00, 0x04,
0x91, 0xa0, 0x40, 0x59, 0x80, 0x00, 0x12, 0x10, 0x00, 0x80, 0x80, 0x65, 0x00,
0x00, 0x04, 0x00, 0x80, 0x40, 0x19, 0x00, 0x21, 0x03, 0x84, 0x60, 0xc0, 0x04,
0x24, 0x1a, 0x12, 0x61, 0x80, 0x80, 0x08, 0x02, 0x04, 0x09, 0x42, 0x12, 0x20,
0x08, 0x34, 0x04, 0x90, 0x20, 0x01, 0x01, 0xa0, 0x00, 0x0b, 0x00, 0x08, 0x91,
0x92, 0x40, 0x02, 0x34, 0x40, 0x88, 0x10, 0x61, 0x19, 0x02, 0x00, 0x40, 0x04,
0x25, 0xc0, 0x80, 0x68, 0x08, 0x04, 0x21, 0x80, 0x22, 0x04, 0x00, 0xa0, 0x0c,
0x01, 0x84, 0x20, 0x41, 0x00, 0x08, 0x8a, 0x00, 0x20, 0x8a, 0x00, 0x48, 0x88,
0x04, 0x04, 0x11, 0x82, 0x08, 0x40, 0x86, 0x09, 0x49, 0xa4, 0x40, 0x00, 0x10,
0x01, 0x01, 0xa2, 0x04, 0x50, 0x80, 0x0c, 0x80, 0x00, 0x48, 0x82, 0xa0, 0x01,
0x18, 0x12, 0x41, 0x01, 0x04, 0x48, 0x41, 0x00, 0x24, 0x01, 0x00, 0x00, 0x88,
0x14, 0x00, 0x02, 0x00, 0x68, 0x01, 0x20, 0x08, 0x4a, 0x22, 0x08, 0x83, 0x80,
0x00, 0x89, 0x04, 0x01, 0xc2, 0x00, 0x00, 0x00, 0x34, 0x04, 0x00, 0x82, 0x28,
0x02, 0x02, 0x41, 0x4a, 0x90, 0x05, 0x82, 0x02, 0x09, 0x80, 0x24, 0x04, 0x41,
0x00, 0x01, 0x92, 0x80, 0x28, 0x01, 0x14, 0x00, 0x50, 0x20, 0x4c, 0x10, 0xb0,
0x04, 0x43, 0xa4, 0x21, 0x90, 0x04, 0x01, 0x02, 0x00, 0x44, 0x48, 0x00, 0x64,
0x08, 0x06, 0x00, 0x42, 0x20, 0x08, 0x02, 0x92, 0x01, 0x4a, 0x00, 0x20, 0x50,
0x32, 0x25, 0x90, 0x22, 0x04, 0x09, 0x00, 0x08, 0x11, 0x80, 0x21, 0x01, 0x10,
0x05, 0x00, 0x32, 0x08, 0x88, 0x94, 0x08, 0x08, 0x24, 0x0d, 0xc1, 0x80, 0x40,
0x0b, 0x20, 0x40, 0x18, 0x12, 0x04, 0x00, 0x22, 0x40, 0x10, 0x26, 0x05, 0xc1,
0x82, 0x00, 0x01, 0x30, 0x24, 0x02, 0x22, 0x41, 0x08, 0x24, 0x48, 0x1a, 0x00,
0x25, 0xd2, 0x12, 0x28, 0x42, 0x00, 0x04, 0x40, 0x30, 0x41, 0x00, 0x02, 0x00,
0x13, 0x20, 0x24, 0xd1, 0x84, 0x08, 0x89, 0x80, 0x04, 0x52, 0x00, 0x44, 0x18,
0xa4, 0x00, 0x00, 0x06, 0x20, 0x91, 0x10, 0x09, 0x42, 0x20, 0x24, 0x40, 0x30,
0x28, 0x00, 0x84, 0x40, 0x40, 0x80, 0x08, 0x10, 0x04, 0x09, 0x08, 0x04, 0x40,
0x08, 0x22, 0x00, 0x19, 0x02, 0x00, 0x00, 0x80, 0x2c, 0x02, 0x02, 0x21, 0x01,
0x90, 0x20, 0x40, 0x00, 0x0c, 0x00, 0x34, 0x48, 0x58, 0x20, 0x01, 0x43, 0x04,
0x20, 0x80, 0x14, 0x00, 0x90, 0x00, 0x6d, 0x11, 0x00, 0x00, 0x40, 0x20, 0x00,
0x03, 0x10, 0x40, 0x88, 0x30, 0x05, 0x4a, 0x00, 0x65, 0x10, 0x24, 0x08, 0x18,
0x84, 0x28, 0x03, 0x80, 0x20, 0x42, 0xb0, 0x40, 0x00, 0x10, 0x69, 0x19, 0x04,
0x00, 0x00, 0x80, 0x04, 0xc2, 0x04, 0x00, 0x01, 0x00, 0x05, 0x00, 0x22, 0x25,
0x08, 0x96, 0x04, 0x02, 0x22, 0x00, 0xd0, 0x10, 0x29, 0x01, 0xa0, 0x60, 0x08,
0x10, 0x04, 0x01, 0x16, 0x44, 0x10, 0x02, 0x28, 0x02, 0x82, 0x48, 0x40, 0x84,
0x20, 0x90, 0x22, 0x28, 0x80, 0x04, 0x00, 0x40, 0x04, 0x24, 0x00, 0x80, 0x29,
0x03, 0x10, 0x60, 0x48, 0x00, 0x00, 0x81, 0xa0, 0x00, 0x51, 0x20, 0x0c, 0xd1,
```

```
0x00, 0x01, 0x41, 0x20, 0x04, 0x92, 0x00, 0x00, 0x10, 0x92, 0x00, 0x42, 0x04,
0x05, 0x01, 0x86, 0x40, 0x80, 0x10, 0x20, 0x52, 0x20, 0x21, 0x00, 0x10, 0x48,
0x0a, 0x02, 0x00, 0xd0, 0x12, 0x41, 0x48, 0x80, 0x04, 0x00, 0x00, 0x48, 0x09,
0x22, 0x04, 0x00, 0x24, 0x00, 0x43, 0x10, 0x60, 0x0a, 0x00, 0x44, 0x12, 0x20,
0x2c, 0x08, 0x20, 0x44, 0x00, 0x84, 0x09, 0x40, 0x06, 0x08, 0xc1, 0x00, 0x40,
0x80, 0x20, 0x00, 0x98, 0x12, 0x48, 0x10, 0xa2, 0x20, 0x00, 0x84, 0x48, 0xc0,
0x10, 0x20, 0x90, 0x12, 0x08, 0x98, 0x82, 0x00, 0x0a, 0xa0, 0x04, 0x03, 0x00,
0x28, 0xc3, 0x00, 0x44, 0x42, 0x10, 0x04, 0x08, 0x04, 0x40, 0x00, 0x00, 0x05,
0x10, 0x00, 0x21, 0x03, 0x80, 0x04, 0x88, 0x12, 0x69, 0x10, 0x00, 0x04, 0x08,
0x04, 0x04, 0x02, 0x84, 0x48, 0x49, 0x04, 0x20, 0x18, 0x02, 0x64, 0x80, 0x30,
0x08, 0x01, 0x02, 0x00, 0x52, 0x12, 0x49, 0x08, 0x20, 0x41, 0x88, 0x10, 0x48,
0x08, 0x34, 0x00, 0x01, 0x86, 0x05, 0xd0, 0x00, 0x00, 0x83, 0x84, 0x21, 0x40,
0x02, 0x41, 0x10, 0x80, 0x48, 0x40, 0xa2, 0x20, 0x51, 0x00, 0x00, 0x49, 0x00,
0x01, 0x90, 0x20, 0x40, 0x18, 0x02, 0x40, 0x02, 0x22, 0x05, 0x40, 0x80, 0x08,
0x82, 0x10, 0x20, 0x18, 0x00, 0x05, 0x01, 0x82, 0x40, 0x58, 0x00, 0x04, 0x81,
0x90, 0x29, 0x01, 0xa0, 0x64, 0x00, 0x22, 0x40, 0x01, 0xa2, 0x00, 0x18, 0x04,
0x0d, 0x00, 0x00, 0x60, 0x80, 0x94, 0x60, 0x82, 0x10, 0x0d, 0x80, 0x30, 0x0c,
0x12, 0x20, 0x00, 0x00, 0x12, 0x40, 0xc0, 0x20, 0x21, 0x58, 0x02, 0x41, 0x10,
0x80, 0x44, 0x03, 0x02, 0x04, 0x13, 0x90, 0x29, 0x08, 0x00, 0x44, 0xc0, 0x00,
0x21, 0x00, 0x26, 0x00, 0x1a, 0x80, 0x01, 0x13, 0x14, 0x20, 0x0a, 0x14, 0x20,
0x00, 0x32, 0x61, 0x08, 0x00, 0x40, 0x42, 0x20, 0x09, 0x80, 0x06, 0x01, 0x81,
0x80, 0x60, 0x42, 0x00, 0x68, 0x90, 0x82, 0x08, 0x42, 0x80, 0x04, 0x02, 0x80,
0x09, 0x0b, 0x04, 0x00, 0x98, 0x00, 0x0c, 0x81, 0x06, 0x44, 0x48, 0x84, 0x28,
0x03, 0x92, 0x00, 0x01, 0x80, 0x40, 0x0a, 0x00, 0x0c, 0x81, 0x02, 0x08, 0x51,
0x04, 0x28, 0x90, 0x02, 0x20, 0x09, 0x10, 0x60, 0x00, 0x00, 0x09, 0x81, 0xa0,
0x0c, 0x00, 0xa4, 0x09, 0x00, 0x02, 0x28, 0x80, 0x20, 0x00, 0x02, 0x02, 0x04,
0x81, 0x14, 0x04, 0x00, 0x04, 0x09, 0x11, 0x12, 0x60, 0x40, 0x20, 0x01, 0x48,
0x30, 0x40, 0x11, 0x00, 0x08, 0x0a, 0x86, 0x00, 0x00, 0x04, 0x60, 0x81, 0x04,
0x01, 0xd0, 0x02, 0x41, 0x18, 0x90, 0x00, 0x0a, 0x20, 0x00, 0xc1, 0x06, 0x01,
0x08, 0x80, 0x64, 0xca, 0x10, 0x04, 0x99, 0x80, 0x48, 0x01, 0x82, 0x20, 0x50,
0x90, 0x48, 0x80, 0x84, 0x20, 0x90, 0x22, 0x00, 0x19, 0x00, 0x04, 0x18, 0x20,
0x24, 0x10, 0x86, 0x40, 0xc2, 0x00, 0x24, 0x12, 0x10, 0x44, 0x00, 0x16, 0x08,
0x10, 0x24, 0x00, 0x12, 0x06, 0x01, 0x08, 0x90, 0x00, 0x12, 0x02, 0x4d, 0x10,
0x80, 0x40, 0x50, 0x22, 0x00, 0x43, 0x10, 0x01, 0x00, 0x30, 0x21, 0x0a, 0x00,
0x00, 0x01, 0x14, 0x00, 0x10, 0x84, 0x04, 0xc1, 0x10, 0x29, 0x0a, 0x00, 0x01,
0x8a, 0x00, 0x20, 0x01, 0x12, 0x0c, 0x49, 0x20, 0x04, 0x81, 0x00, 0x48, 0x01,
0x04, 0x60, 0x80, 0x12, 0x0c, 0x08, 0x10, 0x48, 0x4a, 0x04, 0x28, 0x10, 0x00,
0x28, 0x40, 0x84, 0x45, 0x50, 0x10, 0x60, 0x10, 0x06, 0x44, 0x01, 0x80, 0x09,
0x00, 0x86, 0x01, 0x42, 0xa0, 0x00, 0x90, 0x00, 0x05, 0x90, 0x22, 0x40, 0x41,
0x00, 0x08, 0x80, 0x02, 0x08, 0xc0, 0x00, 0x01, 0x58, 0x30, 0x49, 0x09, 0x14,
0x00, 0x41, 0x02, 0x0c, 0x02, 0x80, 0x40, 0x89, 0x00, 0x24, 0x08, 0x10, 0x05,
0x90, 0x32, 0x40, 0x0a, 0x82, 0x08, 0x00, 0x12, 0x61, 0x00, 0x04, 0x21, 0x00,
0x22, 0x04, 0x10, 0x24, 0x08, 0x0a, 0x04, 0x01, 0x10, 0x00, 0x20, 0x40, 0x84,
0x04, 0x88, 0x22, 0x20, 0x90, 0x12, 0x00, 0x53, 0x06, 0x24, 0x01, 0x04, 0x40,
0x0b, 0x14, 0x60, 0x82, 0x02, 0x0d, 0x10, 0x90, 0x0c, 0x08, 0x20, 0x09, 0x00,
0x14, 0x09, 0x80, 0x80, 0x24, 0x82, 0x00, 0x40, 0x01, 0x02, 0x44, 0x01, 0x20,
0x0c, 0x40, 0x84, 0x40, 0x0a, 0x10, 0x41, 0x00, 0x30, 0x05, 0x09, 0x80, 0x44,
0x08, 0x20, 0x20, 0x02, 0x00, 0x49, 0x43, 0x20, 0x21, 0x00, 0x20, 0x00, 0x01,
0xb6, 0x08, 0x40, 0x04, 0x08, 0x02, 0x80, 0x01, 0x41, 0x80, 0x40, 0x08, 0x10,
0x24, 0x00, 0x20, 0x04, 0x12, 0x86, 0x09, 0xc0, 0x12, 0x21, 0x81, 0x14, 0x04,
0x00, 0x02, 0x20, 0x89, 0xb4, 0x44, 0x12, 0x80, 0x00, 0xd1, 0x00, 0x69, 0x40,
0x80, 0x00, 0x42, 0x12, 0x00, 0x18, 0x04, 0x00, 0x49, 0x06, 0x21, 0x02, 0x04,
0x28, 0x02, 0x84, 0x01, 0xc0, 0x10, 0x68, 0x00, 0x20, 0x08, 0x40, 0x00, 0x08,
0x91, 0x10, 0x01, 0x81, 0x24, 0x04, 0xd2, 0x10, 0x4c, 0x88, 0x86, 0x00, 0x10,
0x80, 0x0c, 0x02, 0x14, 0x00, 0x8a, 0x90, 0x40, 0x18, 0x20, 0x21, 0x80, 0xa4,
0x00, 0x58, 0x24, 0x20, 0x10, 0x10, 0x60, 0xc1, 0x30, 0x41, 0x48, 0x02, 0x48,
0x09, 0x00, 0x40, 0x09, 0x02, 0x05, 0x11, 0x82, 0x20, 0x4a, 0x20, 0x24, 0x18,
0x02, 0x0c, 0x10, 0x22, 0x0c, 0x0a, 0x04, 0x00, 0x03, 0x06, 0x48, 0x48, 0x04,
0x04, 0x02, 0x00, 0x21, 0x80, 0x84, 0x00, 0x18, 0x00, 0x0c, 0x02, 0x12, 0x01,
0x00, 0x14, 0x05, 0x82, 0x10, 0x41, 0x89, 0x12, 0x08, 0x40, 0xa4, 0x21, 0x01,
0x84, 0x48, 0x02, 0x10, 0x60, 0x40, 0x02, 0x28, 0x00, 0x14, 0x08, 0x40, 0xa0,
0x20, 0x51, 0x12, 0x00, 0xc2, 0x00, 0x01, 0x1a, 0x30, 0x40, 0x89, 0x12, 0x4c,
0x02, 0x80, 0x00, 0x00, 0x14, 0x01, 0x01, 0xa0, 0x21, 0x18, 0x22, 0x21, 0x18,
0x06, 0x40, 0x01, 0x80, 0x00, 0x90, 0x04, 0x48, 0x02, 0x30, 0x04, 0x08, 0x00,
0x05, 0x88, 0x24, 0x08, 0x48, 0x04, 0x24, 0x02, 0x06, 0x00, 0x80, 0x00, 0x00,
0x00, 0x10, 0x65, 0x11, 0x90, 0x00, 0x0a, 0x82, 0x04, 0xc3, 0x04, 0x60, 0x48,
```

```
    0x24, 0x04, 0x92, 0x02, 0x44, 0x88, 0x80, 0x40, 0x18, 0x06, 0x29, 0x80, 0x10,
    0x01, 0x00, 0x00, 0x44, 0xc8, 0x10, 0x21, 0x89, 0x30, 0x00, 0x4b, 0xa0, 0x01,
    0x10, 0x14, 0x00, 0x02, 0x94, 0x40, 0x00, 0x20, 0x65, 0x00, 0xa2, 0x0c, 0x40,
    0x22, 0x20, 0x81, 0x12, 0x20, 0x82, 0x04, 0x01, 0x10, 0x00, 0x08, 0x88, 0x00,
    0x00, 0x11, 0x80, 0x04, 0x42, 0x80, 0x40, 0x41, 0x14, 0x00, 0x40, 0x32, 0x2c,
    0x80, 0x24, 0x04, 0x19, 0x00, 0x00, 0x91, 0x00, 0x20, 0x83, 0x00, 0x05, 0x40,
    0x20, 0x09, 0x01, 0x84, 0x40, 0x40, 0x20, 0x20, 0x11, 0x00, 0x40, 0x41, 0x90,
    0x20, 0x00, 0x00, 0x40, 0x90, 0x92, 0x48, 0x18, 0x06, 0x08, 0x81, 0x80, 0x48,
    0x01, 0x34, 0x24, 0x10, 0x20, 0x04, 0x00, 0x20, 0x04, 0x18, 0x06, 0x2d, 0x90,
    0x10, 0x01, 0x00, 0x90, 0x00, 0x0a, 0x22, 0x01, 0x00, 0x22, 0x00, 0x11, 0x84,
    0x01, 0x01, 0x00, 0x20, 0x88, 0x00, 0x44, 0x00, 0x22, 0x01, 0x00, 0xa6, 0x40,
    0x02, 0x06, 0x20, 0x11, 0x00, 0x01, 0xc8, 0xa0, 0x04, 0x8a, 0x00, 0x28, 0x19,
    0x80, 0x00, 0x52, 0xa0, 0x24, 0x12, 0x12, 0x09, 0x08, 0x24, 0x01, 0x48, 0x00,
    0x04, 0x00, 0x24, 0x40, 0x02, 0x84, 0x08, 0x00, 0x04, 0x48, 0x40, 0x90, 0x60,
    0x0a, 0x22, 0x01, 0x88, 0x14, 0x08, 0x01, 0x02, 0x08, 0xd3, 0x00, 0x20, 0xc0,
    0x90, 0x24, 0x10, 0x00, 0x00, 0x01, 0xb0, 0x08, 0x0a, 0xa0, 0x00, 0x80, 0x00,
    0x01, 0x09, 0x00, 0x20, 0x52, 0x02, 0x25, 0x00, 0x24, 0x04, 0x02, 0x84, 0x24,
    0x10, 0x92, 0x40, 0x02, 0xa0, 0x40, 0x00, 0x22, 0x08, 0x11, 0x04, 0x08, 0x01,
    0x22, 0x00, 0x42, 0x14, 0x00, 0x09, 0x90, 0x21, 0x00, 0x30, 0x6c, 0x00, 0x00,
    0x0c, 0x00, 0x22, 0x09, 0x90, 0x10, 0x28, 0x40, 0x00, 0x20, 0xc0, 0x20, 0x00,
    0x90, 0x00, 0x40, 0x01, 0x82, 0x05, 0x12, 0x12, 0x09, 0xc1, 0x04, 0x61, 0x80,
    0x02, 0x28, 0x81, 0x24, 0x00, 0x49, 0x04, 0x08, 0x10, 0x86, 0x29, 0x41, 0x80,
    0x21, 0x0a, 0x30, 0x49, 0x88, 0x90, 0x00, 0x41, 0x04, 0x29, 0x81, 0x80, 0x41,
    0x09, 0x00, 0x40, 0x12, 0x10, 0x40, 0x00, 0x10, 0x40, 0x48, 0x02, 0x05, 0x80,
    0x02, 0x21, 0x40, 0x20, 0x00, 0x58, 0x20, 0x60, 0x00, 0x90, 0x48, 0x00, 0x80,
    0x28, 0xc0, 0x80, 0x48, 0x00, 0x00, 0x44, 0x80, 0x02, 0x00, 0x09, 0x06, 0x00,
    0x12, 0x02, 0x01, 0x00, 0x10, 0x08, 0x83, 0x10, 0x45, 0x12, 0x00, 0x2c, 0x08,
    0x04, 0x44, 0x00, 0x20, 0x20, 0xc0, 0x10, 0x20, 0x01, 0x00, 0x05, 0xc8, 0x20,
    0x04, 0x98, 0x10, 0x08, 0x10, 0x00, 0x24, 0x02, 0x16, 0x40, 0x88, 0x00, 0x61,
    0x88, 0x12, 0x24, 0x80, 0xa6, 0x00, 0x42, 0x00, 0x08, 0x10, 0x06, 0x48, 0x40,
    0xa0, 0x00, 0x50, 0x20, 0x04, 0x81, 0xa4, 0x40, 0x18, 0x00, 0x08, 0x10, 0x80,
    0x01, 0x01};

#if RSA_KEY_SIEVE && SIMULATION && RSA_INSTRUMENT
UINT32 PrimeIndex               = 0;
UINT32 failedAtIteration[10]    = {0};
UINT32 PrimeCounts[3]           = {0};
UINT32 MillerRabinTrials[3]     = {0};
UINT32 totalFieldsSieved[3]     = {0};
UINT32 bitsInFieldAfterSieve[3] = {0};
UINT32 emptyFieldsSieved[3]     = {0};
UINT32 noPrimeFields[3]         = {0};
UINT32 primesChecked[3]         = {0};
UINT16 lastSievePrime           = 0;
#endif
```

7.153 /tpm/src/crypt/RsaKeyCache.c

```
//** Introduction
// This file contains the functions to implement the RSA key cache that can be used
// to speed up simulation.
//
// Only one key is created for each supported key size and it is returned whenever
// a key of that size is requested.
//
// If desired, the key cache can be populated from a file. This allows multiple
// TPM to run with the same RSA keys. Also, when doing simulation, the DRBG will
// use preset sequences so it is not too hard to repeat sequences for debug or
// profile or stress.
//
// When the key cache is enabled, a call to CryptRsaGenerateKey() will call the
// GetCachedRsaKey(). If the cache is enabled and populated, then the cached key
// of the requested size is returned. If a key of the requested size is not
// available, the no key is loaded and the requested key will need to be generated.
// If the cache is not populated, the TPM will open a file that has the appropriate
```

```
// name for the type of keys required (CRT or no-CRT). If the file is the right
// size, it is used. If the file doesn't exist or the file does not have the correct
// size, the TMP will populate the cache with new keys of the required size and
// write the cache data to the file so that they will be available the next time.
//
// Currently, if two simulations are being run with TPM's that have different RSA
// key sizes (e.g,, one with 1024 and 2048 and another with 2048 and 3072, then the
// files will not match for the both of them and they will both try to overwrite
// the other's cache file. I may try to do something about this if necessary.

//** Includes, Types, Locals, and Defines

#include "Tpm.h"

#if USE_RSA_KEY_CACHE

#  include <stdio.h>
#  include "RsaKeyCache_fp.h"

#  if CRT_FORMAT_RSA == YES
#    define CACHE_FILE_NAME "RsaKeyCacheCrt.data"
#  else
#    define CACHE_FILE_NAME "RsaKeyCacheNoCrt.data"
#  endif

typedef struct _RSA_KEY_CACHE_
{
    TPM2B_PUBLIC_KEY_RSA  publicModulus;
    TPM2B_PRIVATE_KEY_RSA privateExponent;
} RSA_KEY_CACHE;

// Determine the number of RSA key sizes for the cache
TPMI_RSA_KEY_BITS SupportedRsaKeySizes[] = {
#  if RSA_1024
    1024,
#  endif
#  if RSA_2048
    2048,
#  endif
#  if RSA_3072
    3072,
#  endif
#  if RSA_4096
    4096,
#  endif
    0};

#  define RSA_KEY_CACHE_ENTRIES (RSA_1024 + RSA_2048 + RSA_3072 + RSA_4096)

// The key cache holds one entry for each of the supported key sizes
RSA_KEY_CACHE s_rsaKeyCache[RSA_KEY_CACHE_ENTRIES];
// Indicates if the key cache is loaded. It can be loaded and enabled or disabled.
BOOL s_keyCacheLoaded = 0;

// Indicates if the key cache is enabled
int s_rsaKeyCacheEnabled = FALSE;

//*** RsaKeyCacheControl()
// Used to enable and disable the RSA key cache.
LIB_EXPORT void RsaKeyCacheControl(int state)
{
    s_rsaKeyCacheEnabled = state;
}

//*** InitializeKeyCache()
// This will initialize the key cache and attempt to write it to a file for later
```

```
// use.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
static BOOL InitializeKeyCache(TPMT_PUBLIC*    publicArea,
                               TPMT_SENSITIVE* sensitive,
                               RAND_STATE* rand  // IN: if not NULL, the deterministic
                                                 //     RNG state
)
{
    int          index;
    TPM_KEY_BITS keySave = publicArea->parameters.rsaDetail.keyBits;
    BOOL         OK      = TRUE;
    //
    s_rsaKeyCacheEnabled = FALSE;
    for(index = 0; OK && index < RSA_KEY_CACHE_ENTRIES; index++)
    {
        publicArea->parameters.rsaDetail.keyBits = SupportedRsaKeySizes[index];
        OK = (CryptRsaGenerateKey(publicArea, sensitive, rand) == TPM_RC_SUCCESS);
        if(OK)
        {
            s_rsaKeyCache[index].publicModulus   = publicArea->unique.rsa;
            s_rsaKeyCache[index].privateExponent = sensitive->sensitive.rsa;
        }
    }
    publicArea->parameters.rsaDetail.keyBits = keySave;
    s_keyCacheLoaded                         = OK;
#  if SIMULATION && USE_RSA_KEY_CACHE && USE_KEY_CACHE_FILE
    if(OK)
    {
        FILE*       cacheFile;
        const char* fn = CACHE_FILE_NAME;

#    if defined     _MSC_VER
        if(fopen_s(&cacheFile, fn, "w+b") != 0)
#    else
        cacheFile = fopen(fn, "w+b");
        if(NULL == cacheFile)
#    endif
        {
            printf("Can't open %s for write.\n", fn);
        }
        else
        {
            fseek(cacheFile, 0, SEEK_SET);
            if(fwrite(s_rsaKeyCache, 1, sizeof(s_rsaKeyCache), cacheFile)
               != sizeof(s_rsaKeyCache))
            {
                printf("Error writing cache to %s.", fn);
            }
        }
        if(cacheFile)
            fclose(cacheFile);
    }
#  endif
    return s_keyCacheLoaded;
}

//*** KeyCacheLoaded()
// Checks that key cache is loaded.
//  Return Type: BOOL
//      TRUE(1)         cache loaded
//      FALSE(0)        cache not loaded
static BOOL KeyCacheLoaded(TPMT_PUBLIC*    publicArea,
                           TPMT_SENSITIVE* sensitive,
                           RAND_STATE*     rand  // IN: if not NULL, the deterministic
```

```
                                        //     RNG state
)
{
#  if SIMULATION && USE_RSA_KEY_CACHE && USE_KEY_CACHE_FILE
    if(!s_keyCacheLoaded)
    {
        FILE*       cacheFile;
        const char* fn = CACHE_FILE_NAME;
#    if defined     _MSC_VER && 1
        if(fopen_s(&cacheFile, fn, "r+b") == 0)
#    else
        cacheFile = fopen(fn, "r+b");
        if(NULL != cacheFile)
#    endif
        {
            fseek(cacheFile, 0L, SEEK_END);
            if(ftell(cacheFile) == sizeof(s_rsaKeyCache))
            {
                fseek(cacheFile, 0L, SEEK_SET);
                s_keyCacheLoaded =
                    (fread(&s_rsaKeyCache, 1, sizeof(s_rsaKeyCache), cacheFile)
                     == sizeof(s_rsaKeyCache));
            }
            fclose(cacheFile);
        }
    }
#  endif
    if(!s_keyCacheLoaded)
        s_rsaKeyCacheEnabled = InitializeKeyCache(publicArea, sensitive, rand);
    return s_keyCacheLoaded;
}

//*** GetCachedRsaKey()
//  Return Type: BOOL
//      TRUE(1)         key loaded
//      FALSE(0)        key not loaded
BOOL GetCachedRsaKey(TPMT_PUBLIC*    publicArea,
                     TPMT_SENSITIVE* sensitive,
                     RAND_STATE*     rand  // IN: if not NULL, the deterministic
                                           //     RNG state
)
{
    int keyBits = publicArea->parameters.rsaDetail.keyBits;
    int index;
    //
    if(KeyCacheLoaded(publicArea, sensitive, rand))
    {
        for(index = 0; index < RSA_KEY_CACHE_ENTRIES; index++)
        {
            if((s_rsaKeyCache[index].publicModulus.t.size * 8) == keyBits)
            {
                publicArea->unique.rsa   = s_rsaKeyCache[index].publicModulus;
                sensitive->sensitive.rsa = s_rsaKeyCache[index].privateExponent;
                return TRUE;
            }
        }
        return FALSE;
    }
    return s_keyCacheLoaded;
}
#endif  // defined SIMULATION && defined USE_RSA_KEY_CACHE
```

7.154 /tpm/src/crypt/Ticket.c

```
//** Introduction
```

```
/*
    This clause contains the functions used for ticket computations.
*/

//** Includes
#include "Tpm.h"
#include "Marshal.h"

//** Functions

//*** TicketIsSafe()
// This function indicates if producing a ticket is safe.
// It checks if the leading bytes of an input buffer is TPM_GENERATED_VALUE
// or its substring of canonical form.  If so, it is not safe to produce ticket
// for an input buffer claiming to be TPM generated buffer
//  Return Type: BOOL
//      TRUE(1)         safe to produce ticket
//      FALSE(0)        not safe to produce ticket
BOOL TicketIsSafe(TPM2B* buffer)
{
    TPM_CONSTANTS32 valueToCompare = TPM_GENERATED_VALUE;
    BYTE            bufferToCompare[sizeof(valueToCompare)];
    BYTE*           marshalBuffer;
    //
    // If the buffer size is less than the size of TPM_GENERATED_VALUE, assume
    // it is not safe to generate a ticket
    if(buffer->size < sizeof(valueToCompare))
        return FALSE;
    marshalBuffer = bufferToCompare;
    TPM_CONSTANTS32_Marshal(&valueToCompare, &marshalBuffer, NULL);
    if(MemoryEqual(buffer->buffer, bufferToCompare, sizeof(valueToCompare)))
        return FALSE;
    else
        return TRUE;
}

//*** TicketComputeVerified()
// This function creates a TPMT_TK_VERIFIED ticket.
/*(See part 2 specification)
//  The ticket is computed as:
//      HMAC(proof, (TPM_ST_VERIFIED | digest | keyName))
//  Where:
//      HMAC()              an HMAC using the hash of proof
//      proof               a TPM secret value associated with the hierarchy
//                          associated with keyName
//      TPM_ST_VERIFIED     a value to differentiate the tickets
//      digest              the signed digest
//      keyName             the Name of the key that signed digest
*/
TPM_RC TicketComputeVerified(
    TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy constant for ticket
    TPM2B_DIGEST*     digest,     // IN: digest
    TPM2B_NAME*       keyName,    // IN: name of key that signed the values
    TPMT_TK_VERIFIED* ticket      // OUT: verified ticket
)
{
    TPM_RC      result = TPM_RC_SUCCESS;
    TPM2B_PROOF proof;
    HMAC_STATE  hmacState;
    //
    // Fill in ticket fields
    ticket->tag       = TPM_ST_VERIFIED;
    ticket->hierarchy = hierarchy;
    result            = HierarchyGetProof(hierarchy, &proof);
    if(result != TPM_RC_SUCCESS)
        return result;
```

```
    // Start HMAC using the proof value of the hierarchy as the HMAC key
    ticket->digest.t.size =
        CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG, &proof.b);
    MemorySet(proof.b.buffer, 0, proof.b.size);

    //  TPM_ST_VERIFIED
    CryptDigestUpdateInt(&hmacState, sizeof(TPM_ST), ticket->tag);
    //  digest
    CryptDigestUpdate2B(&hmacState.hashState, &digest->b);
    // key name
    CryptDigestUpdate2B(&hmacState.hashState, &keyName->b);
    // done
    CryptHmacEnd2B(&hmacState, &ticket->digest.b);

    return TPM_RC_SUCCESS;
}

//*** TicketComputeAuth()
// This function creates a TPMT_TK_AUTH ticket.
/*(See part 2 specification)
//  The ticket is computed as:
//      HMAC(proof, (type || timeout || timeEpoch || cpHash
//                          || policyRef || keyName))
//  where:
//      HMAC()      an HMAC using the hash of proof
//      proof       a TPM secret value associated with the hierarchy of the key
//                  associated with keyName.
//      type        a value to differentiate the tickets.  It could be either
//                  TPM_ST_AUTH_SECRET or TPM_ST_AUTH_SIGNED
//      timeout     TPM-specific value indicating when the authorization expires
//      timeEpoch   TPM-specific value indicating the epoch for the timeout
//      cpHash      optional hash (digest only) of the authorized command
//      policyRef   optional reference to a policy value
//      keyName name of the key that signed the authorization
*/
TPM_RC TicketComputeAuth(
    TPM_ST             type,            // IN: the type of ticket.
    TPMI_RH_HIERARCHY hierarchy,        // IN: hierarchy constant for ticket
    UINT64             timeout,         // IN: timeout
    BOOL               expiresOnReset,  // IN: flag to indicate if ticket expires on
                                        //      TPM Reset
    TPM2B_DIGEST* cpHashA,              // IN: input cpHashA
    TPM2B_NONCE*  policyRef,            // IN: input policyRef
    TPM2B_NAME*   entityName,           // IN: name of entity
    TPMT_TK_AUTH* ticket                // OUT: Created ticket
)
{
    TPM_RC      result = TPM_RC_SUCCESS;
    TPM2B_PROOF proof;
    HMAC_STATE  hmacState;
    //
    // Get proper proof
    result = HierarchyGetProof(hierarchy, &proof);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Fill in ticket fields
    ticket->tag       = type;
    ticket->hierarchy = hierarchy;

    // Start HMAC with hierarchy proof as the HMAC key
    ticket->digest.t.size =
        CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG, &proof.b);
    MemorySet(proof.b.buffer, 0, proof.b.size);
```

```
    //  TPM_ST_AUTH_SECRET or TPM_ST_AUTH_SIGNED,
    CryptDigestUpdateInt(&hmacState, sizeof(UINT16), ticket->tag);
    // cpHash
    CryptDigestUpdate2B(&hmacState.hashState, &cpHashA->b);
    //  policyRef
    CryptDigestUpdate2B(&hmacState.hashState, &policyRef->b);
    //  keyName
    CryptDigestUpdate2B(&hmacState.hashState, &entityName->b);
    //  timeout
    CryptDigestUpdateInt(&hmacState, sizeof(timeout), timeout);
    if(timeout != 0)
    {
        //  epoch
        CryptDigestUpdateInt(&hmacState.hashState, sizeof(CLOCK_NONCE), g_timeEpoch);
        // reset count
        if(expiresOnReset)
            CryptDigestUpdateInt(
                &hmacState.hashState, sizeof(gp.totalResetCount), gp.totalResetCount);
    }
    // done
    CryptHmacEnd2B(&hmacState, &ticket->digest.b);

    return TPM_RC_SUCCESS;
}

//*** TicketComputeHashCheck()
// This function creates a TPMT_TK_HASHCHECK ticket.
/*(See part 2 specification)
//  The ticket is computed as:
//      HMAC(proof, (TPM_ST_HASHCHECK || digest ))
//  where:
//      HMAC()  an HMAC using the hash of proof
//      proof   a TPM secret value associated with the hierarchy
//      TPM_ST_HASHCHECK
//              a value to differentiate the tickets
//      digest  the digest of the data
*/
TPM_RC TicketComputeHashCheck(
    TPMI_RH_HIERARCHY  hierarchy,  // IN: hierarchy constant for ticket
    TPM_ALG_ID         hashAlg,    // IN: the hash algorithm for 'digest'
    TPM2B_DIGEST*      digest,     // IN: input digest
    TPMT_TK_HASHCHECK* ticket      // OUT: Created ticket
)
{
    TPM_RC      result = TPM_RC_SUCCESS;
    TPM2B_PROOF proof;
    HMAC_STATE  hmacState;
    //
    // Get proper proof
    result = HierarchyGetProof(hierarchy, &proof);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Fill in ticket fields
    ticket->tag       = TPM_ST_HASHCHECK;
    ticket->hierarchy = hierarchy;

    // Start HMAC using hierarchy proof as HMAC key
    ticket->digest.t.size =
        CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG, &proof.b);
    MemorySet(proof.b.buffer, 0, proof.b.size);

    //  TPM_ST_HASHCHECK
    CryptDigestUpdateInt(&hmacState, sizeof(TPM_ST), ticket->tag);
    //  hash algorithm
    CryptDigestUpdateInt(&hmacState, sizeof(hashAlg), hashAlg);
```

```
    //  digest
    CryptDigestUpdate2B(&hmacState.hashState, &digest->b);
    // done
    CryptHmacEnd2B(&hmacState, &ticket->digest.b);

    return TPM_RC_SUCCESS;
}

//*** TicketComputeCreation()
// This function creates a TPMT_TK_CREATION ticket.
/*(See part 2 specification)
// The ticket is computed as:
//      HMAC(proof, (TPM_ST_CREATION || Name || hash(TPMS_CREATION_DATA)))
//  Where:
//  HMAC()  an HMAC using the hash of proof
//  proof   a TPM secret value associated with the hierarchy associated with Name
//  TPM_ST_VERIFIED     a value to differentiate the tickets
//  Name    the Name of the object to which the creation data is to be associated
//  TPMS_CREATION_DATA  the creation data structure associated with Name
*/
TPM_RC TicketComputeCreation(TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy for ticket
                             TPM2B_NAME*       name,       // IN: object name
                             TPM2B_DIGEST*     creation,   // IN: creation hash
                             TPMT_TK_CREATION* ticket      // OUT: created ticket
)
{
    TPM_RC      result = TPM_RC_SUCCESS;
    TPM2B_PROOF proof;
    HMAC_STATE  hmacState;

    // Get proper proof
    result = HierarchyGetProof(hierarchy, &proof);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Fill in ticket fields
    ticket->tag       = TPM_ST_CREATION;
    ticket->hierarchy = hierarchy;

    // Start HMAC using hierarchy proof as HMAC key
    ticket->digest.t.size =
        CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG, &proof.b);
    MemorySet(proof.b.buffer, 0, proof.b.size);

    //  TPM_ST_CREATION
    CryptDigestUpdateInt(&hmacState, sizeof(TPM_ST), ticket->tag);
    //  name if provided
    if(name != NULL)
        CryptDigestUpdate2B(&hmacState.hashState, &name->b);
    //  creation hash
    CryptDigestUpdate2B(&hmacState.hashState, &creation->b);
    // Done
    CryptHmacEnd2B(&hmacState, &ticket->digest.b);

    return TPM_RC_SUCCESS;
}
```

7.155 /tpm/src/crypt/ecc/TpmEcc_Signature_ECDAA.c

```
#include "Tpm.h"
#include "TpmEcc_Signature_ECDAA_fp.h"
#include "TpmEcc_Signature_Util_fp.h"
#include "TpmMath_Debug_fp.h"
#include "TpmMath_Util_fp.h"
```

```
#if ALG_ECC && ALG_ECDAA

//*** TpmEcc_SignEcdaa()
//
// This function performs 's' = 'r' + 'T' * 'd' mod 'q' where
// 1) 'r' is a random, or pseudo-random value created in the commit phase
// 2) 'nonceK' is a TPM-generated, random value 0 < 'nonceK' < 'n'
// 3) 'T' is mod 'q' of "Hash"('nonceK' || 'digest'), and
// 4) 'd' is a private key.
//
// The signature is the tuple ('nonceK', 's')
//
// Regrettably, the parameters in this function kind of collide with the parameter
// names used in ECSCHNORR making for a lot of confusion.
//  Return Type: TPM_RC
//      TPM_RC_SCHEME       unsupported hash algorithm
//      TPM_RC_NO_RESULT    cannot get values from random number generator
TPM_RC TpmEcc_SignEcdaa(
    TPM2B_ECC_PARAMETER*  nonceK,  // OUT: 'nonce' component of the signature
    Crypt_Int*            bnS,     // OUT: 's' component of the signature
    const Crypt_EccCurve* E,       // IN: the curve used in signing
    Crypt_Int*            bnD,     // IN: the private key
    const TPM2B_DIGEST*   digest,  // IN: the value to sign (mod 'q')
    TPMT_ECC_SCHEME*      scheme,  // IN: signing scheme (contains the
                                   //      commit count value).
    OBJECT*     eccKey,            // IN: The signing key
    RAND_STATE* rand               // IN: a random number state
)
{
    TPM_RC              retVal;
    TPM2B_ECC_PARAMETER r;
    HASH_STATE          state;
    TPM2B_DIGEST        T;
    CRYPT_INT_MAX(bnT);
    //
    NOT_REFERENCED(rand);
    if(!CryptGenerateR(&r,
                       &scheme->details.ecdaa.count,
                       eccKey->publicArea.parameters.eccDetail.curveID,
                       &eccKey->name))
        retVal = TPM_RC_VALUE;
    else
    {
        // This allocation is here because 'r' doesn't have a value until
        // CrypGenerateR() is done.
        CRYPT_ECC_INITIALIZED(bnR, &r);
        do
        {
            // generate nonceK such that 0 < nonceK < n
            // use bnT as a temp.
            if(!TpmEcc_GenPrivateScalar(bnT, E, rand))
            {
                retVal = TPM_RC_NO_RESULT;
                break;
            }
            TpmMath_IntTo2B(bnT, &nonceK->b, 0);

            T.t.size = CryptHashStart(&state, scheme->details.ecdaa.hashAlg);
            if(T.t.size == 0)
            {
                retVal = TPM_RC_SCHEME;
            }
            else
            {
                CryptDigestUpdate2B(&state, &nonceK->b);
                CryptDigestUpdate2B(&state, &digest->b);
```

```
                CryptHashEnd2B(&state, &T.b);
                TpmMath_IntFrom2B(bnT, &T.b);
                // Watch out for the name collisions in this call!!
                retVal = TpmEcc_SchnorrCalculateS(
                    bnS,
                    bnR,
                    bnT,
                    bnD,
                    ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E)));
            }
        } while(retVal == TPM_RC_NO_RESULT);
        // Because the rule is that internal state is not modified if the command
        // fails, only end the commit if the command succeeds.
        // NOTE that if the result of the Schnorr computation was zero
        // it will probably not be worthwhile to run the same command again because
        // the result will still be zero. This means that the Commit command will
        // need to be run again to get a new commit value for the signature.
        if(retVal == TPM_RC_SUCCESS)
            CryptEndCommit(scheme->details.ecdaa.count);
    }
    return retVal;
}

#endif  // ALG_ECC && ALG_ECDAA
```

7.156 /tpm/src/crypt/ecc/TpmEcc_Signature_ECDSA.c

```
#include "Tpm.h"
#include "TpmEcc_Signature_ECDSA_fp.h"
#include "TpmMath_Debug_fp.h"
#include "TpmMath_Util_fp.h"

#if ALG_ECC && ALG_ECDSA
//*** TpmEcc_AdjustEcdsaDigest()
// Function to adjust the digest so that it is no larger than the order of the
// curve. This is used for ECDSA sign and verification.
static Crypt_Int* TpmEcc_AdjustEcdsaDigest(
    Crypt_Int*          bnD,     // OUT: the adjusted digest
    const TPM2B_DIGEST* digest,  // IN: digest to adjust
    const Crypt_Int*    max      // IN: value that indicates the maximum
                                 //     number of bits in the results
)
{
    int bitsInMax = ExtMath_SizeInBits(max);
    int shift;
    //
    if(digest == NULL)
        ExtMath_SetWord(bnD, 0);
    else
    {
        ExtMath_IntFromBytes(bnD,
                             digest->t.buffer,
                             (NUMBYTES)MIN(digest->t.size, BITS_TO_BYTES(bitsInMax)));
        shift = ExtMath_SizeInBits(bnD) - bitsInMax;
        if(shift > 0)
            ExtMath_ShiftRight(bnD, bnD, shift);
    }
    return bnD;
}

//*** TpmEcc_SignEcdsa()
// This function implements the ECDSA signing algorithm. The method is described
// in the comments below.
TPM_RC
TpmEcc_SignEcdsa(Crypt_Int*            bnR,   // OUT: 'r' component of the signature
```

```
                Crypt_Int*            bnS,    // OUT: 's' component of the signature
                const Crypt_EccCurve* E,      // IN: the curve used in the signature
                                              //     process
                Crypt_Int*            bnD,    // IN: private signing key
                const TPM2B_DIGEST*   digest, // IN: the digest to sign
                RAND_STATE*           rand    // IN: used in debug of signing
)
{
    CRYPT_ECC_NUM(bnK);
    CRYPT_ECC_NUM(bnIk);
    CRYPT_INT_VAR(bnE, MAX_ECC_KEY_BITS);
    CRYPT_POINT_VAR(ecR);
    CRYPT_ECC_NUM(bnX);
    const Crypt_Int* order  = ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E));
    TPM_RC           retVal = TPM_RC_SUCCESS;
    INT32            tries  = 10;
    BOOL             OK     = FALSE;
    //
    pAssert(digest != NULL);
    // The algorithm as described in "Suite B Implementer's Guide to FIPS
    // 186-3(ECDSA)"
    // 1. Use one of the routines in Appendix A.2 to generate (k, k^-1), a
    //    per-message secret number and its inverse modulo n. Since n is prime,
    //    the output will be invalid only if there is a failure in the RBG.
    // 2. Compute the elliptic curve point R = [k]G = (xR, yR) using EC scalar
    //    multiplication (see [Routines]), where G is the base point included in
    //    the set of domain parameters.
    // 3. Compute r = xR mod n. If r = 0, then return to Step 1. 1.
    // 4. Use the selected hash function to compute H = Hash(M).
    // 5. Convert the bit string H to an integer e as described in Appendix B.2.
    // 6. Compute s = (k^-1 *  (e + d *  r)) mod q. If s = 0, return to Step 1.2.
    // 7. Return (r, s).
    // In the code below, q is n (that it, the order of the curve is p)

    do  // This implements the loop at step 6. If s is zero, start over.
    {
        for(; tries > 0; tries--)
        {
            // Step 1 and 2 -- generate an ephemeral key and the modular inverse
            // of the private key.
            if(!TpmEcc_GenerateKeyPair(bnK, ecR, E, rand))
                continue;
            // get mutable copy of X coordinate
            ExtMath_Copy(bnX, ExtEcc_PointX(ecR));
            // x coordinate is mod p.  Make it mod q
            ExtMath_Mod(bnX, order);
            // Make sure that it is not zero;
            if(ExtMath_IsZero(bnX))
                continue;
            // write the modular reduced version of r as part of the signature
            ExtMath_Copy(bnR, bnX);
            // Make sure that a modular inverse exists and try again if not
            OK = (ExtMath_ModInverse(bnIk, bnK, order));
            if(OK)
                break;
        }
        if(!OK)
            goto Exit;

        TpmEcc_AdjustEcdsaDigest(bnE, digest, order);

        // now have inverse of K (bnIk), e (bnE), r (bnR),  d (bnD) and
        // ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E))
        // Compute s = k^-1 (e + r*d)(mod q)
        //  first do s = r*d mod q
        ExtMath_ModMult(bnS, bnR, bnD, order);
```

```
        // s = e + s = e + r * d
        ExtMath_Add(bnS, bnE, bnS);
        // s = k^(-1)s (mod n) = k^(-1)(e + r * d)(mod n)
        ExtMath_ModMult(bnS, bnIk, bnS, order);

        // If S is zero, try again
    } while(ExtMath_IsZero(bnS));
Exit:
    return retVal;
}

//*** TpmEcc_ValidateSignatureEcdsa()
// This function validates an ECDSA signature. rIn and sIn should have been checked
// to make sure that they are in the range 0 < 'v' < 'n'
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE           signature not valid
TPM_RC
TpmEcc_ValidateSignatureEcdsa(
    Crypt_Int*            bnR,  // IN: 'r' component of the signature
    Crypt_Int*            bnS,  // IN: 's' component of the signature
    const Crypt_EccCurve* E,    // IN: the curve used in the signature
                                //     process
    const Crypt_Point*  ecQ,    // IN: the public point of the key
    const TPM2B_DIGEST* digest  // IN: the digest that was signed
)
{
    // Make sure that the allocation for the digest is big enough for a maximum
    // digest
    CRYPT_INT_VAR(bnE, MAX_ECC_KEY_BITS);
    CRYPT_POINT_VAR(ecR);
    CRYPT_ECC_NUM(bnU1);
    CRYPT_ECC_NUM(bnU2);
    CRYPT_ECC_NUM(bnW);
    CRYPT_ECC_NUM(bnV);
    const Crypt_Int* order  = ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E));
    TPM_RC           retVal = TPM_RC_SIGNATURE;
    //
    // Get adjusted digest
    TpmEcc_AdjustEcdsaDigest(bnE, digest, order);
    // 1. If r and s are not both integers in the interval [1, n - 1], output
    //    INVALID.
    //  bnR  and bnS were validated by the caller
    // 2. Use the selected hash function to compute H0 = Hash(M0).
    // This is an input parameter
    // 3. Convert the bit string H0 to an integer e as described in Appendix B.2.
    // Done at entry
    // 4. Compute w = (s')^-1 mod n, using the routine in Appendix B.1.
    if(!ExtMath_ModInverse(bnW, bnS, order))
        goto Exit;
    // 5. Compute u1 = (e' *   w) mod n, and compute u2 = (r' *  w) mod n.
    ExtMath_ModMult(bnU1, bnE, bnW, order);
    ExtMath_ModMult(bnU2, bnR, bnW, order);
    // 6. Compute the elliptic curve point R = (xR, yR) = u1G+u2Q, using EC
    //    scalar multiplication and EC addition (see [Routines]). If R is equal to
    //    the point at infinity O, output INVALID.
    if(TpmEcc_PointMult(
           ecR, ExtEcc_CurveGetG(ExtEcc_CurveGetCurveId(E)), bnU1, ecQ, bnU2, E)
       != TPM_RC_SUCCESS)
        goto Exit;
    // 7. Compute v = Rx mod n.
    ExtMath_Copy(bnV, ExtEcc_PointX(ecR));
    ExtMath_Mod(bnV, order);
    // 8. Compare v and r0. If v = r0, output VALID; otherwise, output INVALID
    if(ExtMath_UnsignedCmp(bnV, bnR) != 0)
        goto Exit;
```

```
    retVal = TPM_RC_SUCCESS;
Exit:
    return retVal;
}

#endif  // ALG_ECC && ALG_ECDSA
```

7.157 /tpm/src/crypt/ecc/TpmEcc_Signature_Schnorr.c

```
#include "Tpm.h"
#include "TpmEcc_Signature_Schnorr_fp.h"
#include "TpmEcc_Signature_Util_fp.h"
#include "TpmMath_Debug_fp.h"
#include "TpmMath_Util_fp.h"

#if ALG_ECC && ALG_ECSCHNORR

//*** SchnorrReduce()
// Function to reduce a hash result if it's magnitude is too large. The size of
// 'number' is set so that it has no more bytes of significance than 'reference'
// value. If the resulting number can have more bits of significance than
// 'reference'.
static void SchnorrReduce(TPM2B*           number,    // IN/OUT: Value to reduce
                          const Crypt_Int* reference  // IN: the reference value
)
{
    UINT16 maxBytes = (UINT16)BITS_TO_BYTES(ExtMath_SizeInBits(reference));
    if(number->size > maxBytes)
        number->size = maxBytes;
}

//*** SchnorrEcc()
// This function is used to perform a modified Schnorr signature.
//
// This function will generate a random value 'k' and compute
// a) ('xR', 'yR') = ['k']'G'
// b) 'r' = "Hash"('xR' || 'P')(mod 'q')
// c) 'rT' = truncated 'r'
// d) 's'= 'k' + 'rT' * 'ds' (mod 'q')
// e) return the tuple 'rT', 's'
//
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        failure in the Schnorr sign process
//      TPM_RC_SCHEME           hashAlg can't produce zero-length digest
TPM_RC TpmEcc_SignEcSchnorr(
    Crypt_Int*            bnR,      // OUT: 'r' component of the signature
    Crypt_Int*            bnS,      // OUT: 's' component of the signature
    const Crypt_EccCurve* E,        // IN: the curve used in signing
    Crypt_Int*            bnD,      // IN: the signing key
    const TPM2B_DIGEST*   digest,   // IN: the digest to sign
    TPM_ALG_ID            hashAlg,  // IN: signing scheme (contains a hash)
    RAND_STATE*           rand      // IN: non-NULL when testing
)
{
    HASH_STATE hashState;
    UINT16     digestSize = CryptHashGetDigestSize(hashAlg);
    TPM2B_TYPE(T, MAX(MAX_DIGEST_SIZE, MAX_ECC_KEY_BYTES));
    TPM2B_T          T2b;
    TPM2B*           e      = &T2b.b;
    TPM_RC           retVal = TPM_RC_NO_RESULT;
    const Crypt_Int* order;
    const Crypt_Int* prime;
    CRYPT_ECC_NUM(bnK);
    CRYPT_POINT_VAR(ecR);
    //
```

```
    // Parameter checks
    if(E == NULL)
        ERROR_EXIT(TPM_RC_VALUE);

    order = ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E));
    prime = ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E));

    // If the digest does not produce a hash, then null the signature and return
    // a failure.
    if(digestSize == 0)
    {
        ExtMath_SetWord(bnR, 0);
        ExtMath_SetWord(bnS, 0);
        ERROR_EXIT(TPM_RC_SCHEME);
    }
    do
    {
        // Generate a random key pair
        if(!TpmEcc_GenerateKeyPair(bnK, ecR, E, rand))
            break;
        // Convert R.x to a string
        TpmMath_IntTo2B(ExtEcc_PointX(ecR),
                        e,
                        (NUMBYTES)BITS_TO_BYTES(ExtMath_SizeInBits(prime)));

        // f) compute r = Hash(e || P) (mod n)
        CryptHashStart(&hashState, hashAlg);
        CryptDigestUpdate2B(&hashState, e);
        CryptDigestUpdate2B(&hashState, &digest->b);
        e->size = CryptHashEnd(&hashState, digestSize, e->buffer);
        // Reduce the hash size if it is larger than the curve order
        SchnorrReduce(e, order);
        // Convert hash to number
        TpmMath_IntFrom2B(bnR, e);
        // Do the Schnorr computation
        retVal = TpmEcc_SchnorrCalculateS(
            bnS, bnK, bnR, bnD, ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E)));
    } while(retVal == TPM_RC_NO_RESULT);
Exit:
    return retVal;
}

//*** TpmEcc_ValidateSignatureEcSchnorr()
// This function is used to validate an EC Schnorr signature.
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE        signature not valid
TPM_RC TpmEcc_ValidateSignatureEcSchnorr(
    Crypt_Int*            bnR,      // IN: 'r' component of the signature
    Crypt_Int*            bnS,      // IN: 's' component of the signature
    TPM_ALG_ID            hashAlg,  // IN: hash algorithm of the signature
    const Crypt_EccCurve* E,        // IN: the curve used in the signature
                                    //     process
    Crypt_Point*        ecQ,        // IN: the public point of the key
    const TPM2B_DIGEST* digest      // IN: the digest that was signed
)
{
    CRYPT_INT_MAX(bnRn);
    CRYPT_POINT_VAR(ecE);
    CRYPT_INT_MAX(bnEx);
    const Crypt_Int* order      = ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E));
    UINT16           digestSize = CryptHashGetDigestSize(hashAlg);
    HASH_STATE       hashState;
    TPM2B_TYPE(BUFFER, MAX(MAX_ECC_PARAMETER_BYTES, MAX_DIGEST_SIZE));
    TPM2B_BUFFER Ex2 = {{sizeof(Ex2.t.buffer), {0}}};
    BOOL         OK;
    //
```

```
    // E = [s]G - [r]Q
    ExtMath_Mod(bnR, order);
    // Make -r = n - r
    ExtMath_Subtract(bnRn, order, bnR);
    // E = [s]G + [-r]Q
    OK = TpmEcc_PointMult(
             ecE, ExtEcc_CurveGetG(ExtEcc_CurveGetCurveId(E)), bnS, ecQ, bnRn, E)
         == TPM_RC_SUCCESS;
    //   // reduce the x portion of E mod q
    //    OK = OK && ExtMath_Mod(ecE->x, order);
    // Convert to byte string
    OK = OK
         && TpmMath_IntTo2B(ExtEcc_PointX(ecE),
                            &Ex2.b,
                            (NUMBYTES)(BITS_TO_BYTES(ExtMath_SizeInBits(order))));
    if(OK)
    {
        // Ex = h(pE.x || digest)
        CryptHashStart(&hashState, hashAlg);
        CryptDigestUpdate(&hashState, Ex2.t.size, Ex2.t.buffer);
        CryptDigestUpdate(&hashState, digest->t.size, digest->t.buffer);
        Ex2.t.size = CryptHashEnd(&hashState, digestSize, Ex2.t.buffer);
        SchnorrReduce(&Ex2.b, order);
        TpmMath_IntFrom2B(bnEx, &Ex2.b);
        // see if Ex matches R
        OK = ExtMath_UnsignedCmp(bnEx, bnR) == 0;
    }
    return (OK) ? TPM_RC_SUCCESS : TPM_RC_SIGNATURE;
}

#endif  // ALG_ECC && ALG_ECSCHNORR
```

7.158 /tpm/src/crypt/ecc/TpmEcc_Signature_SM2.c

```
#include "Tpm.h"
#include "TpmEcc_Signature_SM2_fp.h"
#include "TpmMath_Debug_fp.h"
#include "TpmMath_Util_fp.h"

#if ALG_ECC && ALG_SM2

//*** TpmEcc_SignEcSm2()
// This function signs a digest using the method defined in SM2 Part 2. The method
// in the standard will add a header to the message to be signed that is a hash of
// the values that define the key. This then hashed with the message to produce a
// digest ('e'). This function signs 'e'.
//  Return Type: TPM_RC
//      TPM_RC_VALUE         bad curve
TPM_RC TpmEcc_SignEcSm2(Crypt_Int* bnR,  // OUT: 'r' component of the signature
                        Crypt_Int* bnS,  // OUT: 's' component of the signature
                        const Crypt_EccCurve* E,    // IN: the curve used in signing
                        Crypt_Int*            bnD,  // IN: the private key
                        const TPM2B_DIGEST*   digest,  // IN: the digest to sign
                        RAND_STATE* rand  // IN: random number generator (mostly for
                                          //     debug)
)
{
    CRYPT_INT_MAX_INITIALIZED(bnE, digest);  // Don't know how big digest might be
    CRYPT_ECC_NUM(bnN);
    CRYPT_ECC_NUM(bnK);
    CRYPT_ECC_NUM(bnT);  // temp
    CRYPT_POINT_VAR(Q1);
    const Crypt_Int* order =
        (E != NULL) ? ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E)) : NULL;
//
```

```
#   ifdef _SM2_SIGN_DEBUG
    TpmEccDebug_FromHex(bnE,
                        "B524F552CD82B8B028476E005C377FB1"
                        "9A87E6FC682D48BB5D42E3D9B9EFFE76",
                        MAX_ECC_KEY_BYTES);
    TpmEccDebug_FromHex(bnD,
                        "128B2FA8BD433C6C068C8D803DFF7979"
                        "2A519A55171B1B650C23661D15897263",
                        MAX_ECC_KEY_BYTES);
#   endif
    // A3: Use random number generator to generate random number 1 <= k <= n-1;
    // NOTE: Ax: numbers are from the SM2 standard
loop:
{
    // Get a random number 0 < k < n
    TpmMath_GetRandomInRange(bnK, order, rand);
#   ifdef _SM2_SIGN_DEBUG
    TpmEccDebug_FromHex(bnK,
                        "6CB28D99385C175C94F94E934817663F"
                        "C176D925DD72B727260DBAAE1FB2F96F",
                        MAX_ECC_KEY_BYTES);
#   endif
    // A4: Figure out the point of elliptic curve (x1, y1)=[k]G, and according
    // to details specified in 4.2.7 in Part 1 of this document, transform the
    // data type of x1 into an integer;
    if(!ExtEcc_PointMultiply(Q1, NULL, bnK, E))
        goto loop;
    // A5: Figure out 'r' = ('e' + 'x1') mod 'n',
    ExtMath_Add(bnR, bnE, ExtEcc_PointX(Q1));
    ExtMath_Mod(bnR, order);
#   ifdef _SM2_SIGN_DEBUG
    pAssert(TpmEccDebug_HexEqual(bnR,
                                 "40F1EC59F793D9F49E09DCEF49130D41"
                                 "94F79FB1EED2CAA55BACDB49C4E755D1"));
#   endif
    // if r=0 or r+k=n, return to A3;
    if(ExtMath_IsZero(bnR))
        goto loop;
    ExtMath_Add(bnT, bnK, bnR);
    if(ExtMath_UnsignedCmp(bnT, bnN) == 0)
        goto loop;
    // A6: Figure out s = ((1 + dA)^-1  (k - r  dA)) mod n,
    // if s=0, return to A3;
    // compute t = (1+dA)^-1
    ExtMath_AddWord(bnT, bnD, 1);
    ExtMath_ModInverse(bnT, bnT, order);
#   ifdef _SM2_SIGN_DEBUG
    pAssert(TpmEccDebug_HexEqual(bnT,
                                 "79BFCF3052C80DA7B939E0C6914A18CB"
                                 "B2D96D8555256E83122743A7D4F5F956"));
#   endif
    // compute s = t * (k - r * dA) mod n
    ExtMath_ModMult(bnS, bnR, bnD, order);
    // k - r * dA mod n = k + n - ((r * dA) mod n)
    ExtMath_Subtract(bnS, order, bnS);
    ExtMath_Add(bnS, bnK, bnS);
    ExtMath_ModMult(bnS, bnS, bnT, order);
#   ifdef _SM2_SIGN_DEBUG
    pAssert(TpmEccDebug_HexEqual(bnS,
                                 "6FC6DAC32C5D5CF10C77DFB20F7C2EB6"
                                 "67A457872FB09EC56327A67EC7DEEBE7"));
#   endif
    if(ExtMath_IsZero(bnS))
        goto loop;
}
// A7: According to details specified in 4.2.1 in Part 1 of this document,
```

```
// transform the data type of r, s into bit strings, signature of message M
// is (r, s).
// This is handled by the common return code
#  ifdef _SM2_SIGN_DEBUG
    pAssert(TpmEccDebug_HexEqual(bnR,
                                 "40F1EC59F793D9F49E09DCEF49130D41"
                                 "94F79FB1EED2CAA55BACDB49C4E755D1"));
    pAssert(TpmEccDebug_HexEqual(bnS,
                                 "6FC6DAC32C5D5CF10C77DFB20F7C2EB6"
                                 "67A457872FB09EC56327A67EC7DEEBE7"));
#  endif
    return TPM_RC_SUCCESS;
}

//*** TpmEcc_ValidateSignatureEcSm2()
// This function is used to validate an SM2 signature.
//  Return Type: TPM_RC
//      TPM_RC_SIGNATURE            signature not valid
TPM_RC TpmEcc_ValidateSignatureEcSm2(
    Crypt_Int*            bnR,  // IN: 'r' component of the signature
    Crypt_Int*            bnS,  // IN: 's' component of the signature
    const Crypt_EccCurve* E,    // IN: the curve used in the signature
                                //     process
    Crypt_Point*        ecQ,    // IN: the public point of the key
    const TPM2B_DIGEST* digest  // IN: the digest that was signed
)
{
    CRYPT_POINT_VAR(P);
    CRYPT_ECC_NUM(bnRp);
    CRYPT_ECC_NUM(bnT);
    CRYPT_INT_MAX_INITIALIZED(bnE, digest);
    BOOL             OK;
    const Crypt_Int* order = ExtEcc_CurveGetOrder(ExtEcc_CurveGetCurveId(E));

#  ifdef _SM2_SIGN_DEBUG
    // Make sure that the input signature is the test signature
    pAssert(TpmEccDebug_HexEqual(bnR,
                                 "40F1EC59F793D9F49E09DCEF49130D41"
                                 "94F79FB1EED2CAA55BACDB49C4E755D1"));
    pAssert(TpmEccDebug_HexEqual(bnS,
                                 "6FC6DAC32C5D5CF10C77DFB20F7C2EB6"
                                 "67A457872FB09EC56327A67EC7DEEBE7"));
#  endif
    // b)   compute t  := (r + s) mod n
    ExtMath_Add(bnT, bnR, bnS);
    ExtMath_Mod(bnT, order);
#  ifdef _SM2_SIGN_DEBUG
    pAssert(TpmEccDebug_HexEqual(bnT,
                                 "2B75F07ED7ECE7CCC1C8986B991F441A"
                                 "D324D6D619FE06DD63ED32E0C997C801"));
#  endif
    // c)   verify that t > 0
    OK = !ExtMath_IsZero(bnT);
    if(!OK)
        // set T to a value that should allow rest of the computations to run
        // without trouble
        ExtMath_Copy(bnT, bnS);
    // d)   compute (x, y) := [s]G + [t]Q
    OK = ExtEcc_PointMultiplyAndAdd(P, NULL, bnS, ecQ, bnT, E);
#  ifdef _SM2_SIGN_DEBUG
    pAssert(OK
            && TpmEccDebug_HexEqual(ExtEcc_PointX(P),
                                    "110FCDA57615705D5E7B9324AC4B856D"
                                    "23E6D9188B2AE47759514657CE25D112"));
#  endif
    // e)   compute r' := (e + x) mod n (the x coordinate is in bnT)
```

```
    OK = OK && ExtMath_Add(bnRp, bnE, ExtEcc_PointX(P));
    OK = OK && ExtMath_Mod(bnRp, order);

    // f)   verify that r' = r
    OK = OK && (ExtMath_UnsignedCmp(bnR, bnRp) == 0);

    if(!OK)
        return TPM_RC_SIGNATURE;
    else
        return TPM_RC_SUCCESS;
}

#endif  // ALG_ECC && ALG_SM2
```

7.159 /tpm/src/crypt/ecc/TpmEcc_Signature_Util.c

```
// functions shared by multiple signature algorithms
#include "Tpm.h"
#include "TpmEcc_Signature_Util_fp.h"
#include "TpmMath_Debug_fp.h"
#include "TpmMath_Util_fp.h"

#if(ALG_ECC && (ALG_ECSCHNORR || ALG_ECDAA))

//*** TpmEcc_SchnorrCalculateS()
// This contains the Schnorr signature (S) computation. It is used by both ECDAA and
// Schnorr signing. The result is computed as: ['s' = 'k' + 'r' * 'd' (mod 'n')]
// where
// 1) 's' is the signature
// 2) 'k' is a random value
// 3) 'r' is the value to sign
// 4) 'd' is the private EC key
// 5) 'n' is the order of the curve
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        the result of the operation was zero or 'r' (mod 'n')
//                              is zero
TPM_RC TpmEcc_SchnorrCalculateS(
    Crypt_Int*       bnS,  // OUT: 's' component of the signature
    const Crypt_Int* bnK,  // IN: a random value
    Crypt_Int*       bnR,  // IN: the signature 'r' value
    const Crypt_Int* bnD,  // IN: the private key
    const Crypt_Int* bnN   // IN: the order of the curve
)
{
    // Need a local temp value to store the intermediate computation because product
    // size can be larger than will fit in bnS.
    CRYPT_INT_VAR(bnT1, MAX_ECC_PARAMETER_BYTES * 2 * 8);
    //
    // Reduce bnR without changing the input value
    ExtMath_Divide(NULL, bnT1, bnR, bnN);
    if(ExtMath_IsZero(bnT1))
        return TPM_RC_NO_RESULT;
    // compute s = (k + r * d)(mod n)
    // r * d
    ExtMath_Multiply(bnT1, bnT1, bnD);
    // k + r * d
    ExtMath_Add(bnT1, bnT1, bnK);
    // k + r * d (mod n)
    ExtMath_Divide(NULL, bnS, bnT1, bnN);
    return (ExtMath_IsZero(bnS)) ? TPM_RC_NO_RESULT : TPM_RC_SUCCESS;
}

#endif  // (ALG_ECC && (ALG_ECSCHNORR || ALG_ECDAA))
```

7.160 /tpm/src/crypt/ecc/TpmEcc_Util.c

```
//** Introduction
// This file contains utility functions to help using the external Math library
// for Ecc functions.
#include "Tpm.h"
#include "TpmMath_Util_fp.h"

#if ALG_ECC

//***
// TpmEcc_PointFrom2B() Function to create a Crypt_Point structure from a 2B
// point. The target point is expected to have memory allocated and
// uninitialized. A TPMS_ECC_POINT is going to be two ECC values in the same
// buffer. The values are going to be the size of the modulus. They are in
// modular form.
//
// NOTE: This function considers both parameters optional because of use
// cases where points may not be specified in the calling function. If the
// initializer or point buffer is NULL, then NULL is returned. As a result, the
// only error detection when the initializer value is invalid is to return NULL
// in that error case as well. If a caller wants to handle that error case
// differently, then the caller must perform the correct validation before/after
// this function.
LIB_EXPORT Crypt_Point* TpmEcc_PointFrom2B(
    Crypt_Point*    ecP,  // OUT: the preallocated point structure
    TPMS_ECC_POINT* p     // IN: the number to convert
)
{
    if(p == NULL)
        return NULL;

    if(ecP != NULL)
    {
        return ExtEcc_PointFromBytes(
            ecP, p->x.t.buffer, p->x.t.size, p->y.t.buffer, p->y.t.size);
    }
    return ecP;  // will return NULL if ecP is NULL.
}

//*** TpmEcc_PointTo2B()
// This function converts a BIG_POINT into a TPMS_ECC_POINT. A TPMS_ECC_POINT
// contains two TPM2B_ECC_PARAMETER values. The maximum size of the parameters
// is dependent on the maximum EC key size used in an implementation.
// The presumption is that the TPMS_ECC_POINT is large enough to hold 2 TPM2B
// values, each as large as a MAX_ECC_PARAMETER_BYTES
LIB_EXPORT BOOL TpmEcc_PointTo2B(
    TPMS_ECC_POINT*       p,    // OUT: the converted 2B structure
    const Crypt_Point*    ecP,  // IN: the values to be converted
    const Crypt_EccCurve* E     // IN: curve descriptor for the point
)
{
    pAssert(p && ecP && E);
    TPM_ECC_CURVE curveId = ExtEcc_CurveGetCurveId(E);
    NUMBYTES      size    = CryptEccGetKeySizeForCurve(curveId);
    size                  = (UINT16)BITS_TO_BYTES(size);
    MemorySet(p, 0, sizeof(*p));
    p->x.t.size = size;
    p->y.t.size = size;
    return ExtEcc_PointToBytes(
        ecP, p->x.t.buffer, &p->x.t.size, p->y.t.buffer, &p->y.t.size);
}

#endif  // ALG_ECC
```

7.161 /tpm/src/crypt/math/TpmMath_Debug.c

```
//** Introduction
// This file contains debug utility functions to help testing Ecc.
#include "Tpm.h"
#include "TpmEcc_Util_fp.h"
#include "TpmMath_Debug_fp.h"

#if ALG_SM2
#  ifdef _SM2_SIGN_DEBUG

//*** SafeGetStringLength()
// self-implemented version of strnlen_s.  This is necessary because
// some environments don't have a C-runtime library, or are limited to
// C99, and strnlen_s was standardized in C11.
static size_t SafeGetStringLength(const char* string, size_t maxsize)
{
    // strnlen_s has two boundary conditions:
    // return 0 if pointer is nullptr, or
    // maxsize if no null character is found.
    if(string == NULL)
        return 0;

    const char* pos  = string;
    size_t      size = 0;

    while(*pos != '\0' && size < maxsize)
    {
        pos++;
        size++;
    }
    return size;
}

// convert from hex value.  If invalid, result will be out of range.
static LIB_EXPORT BYTE FromHex(unsigned char c)
{
    // hack for the ASCII characters we care about
    BYTE upper = (c & (~0x20));
    if(c >= '0' && c <= '9')
        return c - '0';
    else if(c >= 'A' && c <= 'F')
        return c - 'A';

    return 255;
}

//*** TpmEccDebug_FromHex()
// Convert a hex string into a Crypt_Int*. This is primarily used in debugging.
LIB_EXPORT Crypt_Int* TpmEccDebug_FromHex(
    Crypt_Int*           bn,         // OUT:
    const unsigned char* hex,        // IN:
    size_t               maxsizeHex  // IN: maximum size of hex
)
{
    // if value is larger than this, then fail
    BYTE tempBuf[MAX_ECC_KEY_BYTES];
    MemorySet(tempBuf, 0, sizeof(tempBuf));
    ExtMath_SetWord(bn, 0);

    size_t len = SafeGetStringLength(hex, maxsizeHex);
    BOOL   OK  = FALSE;
    if((len % 2) == 0)
    {
        OK = TRUE;
        for(size_t i = 0; i < len; i += 2)
```

```
        {
            BYTE highNibble = FromHex(*hex);
            hex++;
            BYTE lowNibble = FromHex(*hex);
            hex++;
            // unsigned, no need to check zero
            if(highNibble > 15 || lowNibble > 15)
            {
                OK = FALSE;
                break;
            }
            BYTE b          = ((highNibble << 4) | lowNibble);
            tempBuf[i / 2] = b;
        }
        if(OK)
        {
            ExtMath_IntFromBytes(bn, tempBuf, (NUMBYTES)(len / 2));
        }
    }

    if(!OK)
    {
        // this should only be called in testing, so any
        // errors are fatal.
        FAIL(FATAL_ERROR_INTERNAL);
    }
    return bn;
}

//*** TpmEccDebug_HexEqual()
// This function compares a bignum value to a hex string.
// using TpmEcc namespace because code assumes the max size
// is correct for ECC.
//  Return Type: BOOL
//      TRUE(1)         values equal
//      FALSE(0)        values not equal
BOOL TpmEccDebug_HexEqual(const Crypt_Int* bn,  //IN: big number value
                          const char*      c    //IN: character string number
)
{
    CRYPT_ECC_NUM(bnC);
    TpmEccDebug_FromHex(bnC, c, MAX_ECC_KEY_BYTES * 2 + 1);
    return (ExtMath_UnsignedCmp(bn, bnC) == 0);
}
#  endif  // _SM2_SIGN_DEBUG
#endif    // ALG_SM2
```

7.162 /tpm/src/crypt/math/TpmMath_Util.c

```
//** Introduction
// This file contains utility functions to help using the external Math library
#include "Tpm.h"
#include "TpmMath_Util_fp.h"

//*** TpmMath_IntFrom2B()
// Convert an TPM2B to a Crypt_Int.
// If the input value does not exist, or the output does not exist, or the input
// will not fit into the output the function returns NULL
LIB_EXPORT Crypt_Int* TpmMath_IntFrom2B(Crypt_Int*   value,  // OUT:
                                        const TPM2B* a2B     // IN: number to convert
)
{
    if(value != NULL && a2B != NULL)
        return ExtMath_IntFromBytes(value, a2B->buffer, a2B->size);
    return NULL;
```

```
}

//*** TpmMath_IntTo2B()
//
// Function to convert a Crypt_Int to TPM2B. The TPM2B bytes are
// always in big-endian ordering (most significant byte first). If 'size' is
// non-zero and less than required by `value` then an error is returned. If
// `size` is non-zero and larger than `value`, the result buffer is padded
// with zeros. If `size` is zero, then the TPM2B is assumed to be large enough
// for the data and a2b->size will be adjusted accordingly.
LIB_EXPORT BOOL TpmMath_IntTo2B(
    const Crypt_Int* value,  // IN: value to convert
    TPM2B*           a2B,    // OUT: buffer for output
    NUMBYTES         size    // IN: Size of output buffer - see comments.
)
{
    // Set the output size
    if(value && a2B)
    {
        a2B->size = size;
        return ExtMath_IntToBytes(value, a2B->buffer, &a2B->size);
    }
    return FALSE;
}

//*** TpmMath_GetRandomBits()
// This function gets random bits for use in various places.
//
// One consequence of the generation scheme is that, if the number of bits requested
// is not a multiple of 8, then the high-order bits are set to zero. This would come
// into play when generating a 521-bit ECC key. A 66-byte (528-bit) value is
// generated and the high order 7 bits are masked off (CLEAR).
// In this situation, the highest order byte is the first byte (big-endian/TPM2B
format)
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
LIB_EXPORT BOOL TpmMath_GetRandomBits(BYTE* pBuffer, size_t bits, RAND_STATE* rand)
{
    // buffer is assumed to be large enough for the number of bits rounded up to
    // bytes.
    NUMBYTES byteCount = (NUMBYTES)BITS_TO_BYTES(bits);
    if(DRBG_Generate(rand, pBuffer, byteCount) == byteCount)
    {
        // now flip the buffer order - this exists only to maintain
        // compatibility with existing Known-value tests that expect the
        // GetRandomInteger behavior of generating the value in little-endian
        // order.
        BYTE* pFrom = pBuffer + byteCount - 1;
        BYTE* pTo   = pBuffer;
        while(pTo < pFrom)
        {
            BYTE t = *pTo;
            *pTo   = *pFrom;
            *pFrom = t;
            pTo++;
            pFrom--;
        }
        // For a little-endian machine, the conversion is a straight byte
        // reversal, done above. For a big-endian machine, we have to put the
        // words in big-endian byte order.  COMPATIBILITY NOTE: This code does
        // not exactly reproduce the original code, because the original big-num
        // code always generated data in units of crypt_word_t sizes.  I.e. you
        // couldn't generate just 9 bits for example.  This revised version of
        // the function could; and would generate 2 bytes with the first byte
        // masked to 1 bit.  In order to avoid running over the buffer when
```

```
        // swapping crypt_uword_t blocks, this loop intentionally doesn't swap
        // the last word if it is smaller than crypt_word_t size (which is the
        // same as saying the buffer isn't an integral number of crypt_word_t
        // units.) This is okay in this particular case _because_ this whole
        // block of swapping code is to maintain compatibilty with existing
        // KNOWN ANSWER TESTS, and said existing tests use sizes that this
        // assumption is true for.  Any new code with a different size where
        // this last partial value isn't swapped will be creating a new KAT, and
        // thus any (cryptographically valid) value is still random; swapping
        // doesn't make a cryptographic random buffer more or less random, so
        // the failure to swap is fine.
#if BIG_ENDIAN_TPM
        crypt_uword_t* pTemp = (crypt_uword_t*)pBuffer;
        for(size_t t = 0; t < (byteCount / sizeof(crypt_uword_t)); t++)
            *pTemp = SWAP_CRYPT_WORD(*pTemp);
#endif
        // if the number of bits % 8 != 0, mask the high order (first) byte to the
relevant number of bits
        // bits % 8     desired mask   right-shift of 0xFF
        //     0           0xFF             0 = (8 - 0) % 8
        //     1           0x01             7 = (8 - 1) % 8
        //     2           0x03             6 = (8 - 2) % 8
        //     ... etc ...
        //     7           0x7F             1 = (8 - 7) % 8
        int  excessBits = bits % 8;
        int  shift      = (8 - excessBits) % 8;
        BYTE mask       = ~(0xFF >> shift);
        pBuffer[0]      = pBuffer[0] & mask;
        return TRUE;
    }
    return FALSE;
}

//*** TpmMath_GetRandomInteger()
// This function gets random bits for use in various places. To make sure that the
// number is generated in a portable format, it is created as a TPM2B and then
// converted to the internal format.
//
// One consequence of the generation scheme is that, if the number of bits requested
// is not a multiple of 8, then the high-order bits are set to zero. This would come
// into play when generating a 521-bit ECC key. A 66-byte (528-bit) value is
// generated an the high order 7 bits are masked off (CLEAR).
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure
LIB_EXPORT BOOL TpmMath_GetRandomInteger(Crypt_Int* n, size_t bits, RAND_STATE* rand)
{
    // Since this could be used for ECC key generation using the extra bits method,
    // make sure that the value is large enough
    TPM2B_TYPE(LARGEST, LARGEST_NUMBER + 8);
    TPM2B_LARGEST large;
    //
    large.b.size = (UINT16)BITS_TO_BYTES(bits);
    if(DRBG_Generate(rand, large.t.buffer, large.t.size) == large.t.size)
    {
        if(TpmMath_IntFrom2B(n, &large.b) != NULL)
        {
            if(ExtMath_MaskBits(n, (crypt_uword_t)bits))
                return TRUE;
        }
    }
    return FALSE;
}

//*** BnGenerateRandomInRange()
// This function is used to generate a random number r in the range 1 <= r < limit.
```

```
// The function gets a random number of bits that is the size of limit. There is some
// some probability that the returned number is going to be greater than or equal
// to the limit. If it is, try again. There is no more than 50% chance that the
// next number is also greater, so try again. We keep trying until we get a
// value that meets the criteria. Since limit is very often a number with a LOT of
// high order ones, this rarely would need a second try.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure ('limit' is too small)
LIB_EXPORT BOOL TpmMath_GetRandomInRange(
    Crypt_Int* dest, const Crypt_Int* limit, RAND_STATE* rand)
{
    size_t bits = ExtMath_SizeInBits(limit);
    //
    if(bits < 2)
    {
        ExtMath_SetWord(dest, 0);
        return FALSE;
    }
    else
    {
        while(TpmMath_GetRandomInteger(dest, bits, rand)
              && (ExtMath_IsZero(dest) || (ExtMath_UnsignedCmp(dest, limit) >= 0)))
            ;
    }
    return !g_inFailureMode;
}
```

7.163 /tpm/src/events/_TPM_Hash_Data.c

```
#include "Tpm.h"

// This function is called to process a _TPM_Hash_Data indication.
LIB_EXPORT void _TPM_Hash_Data(uint32_t dataSize,   // IN: size of data to be extend
                               unsigned char* data  // IN: data buffer
)
{
    UINT32       i;
    HASH_OBJECT* hashObject;
    TPMI_DH_PCR  pcrHandle = TPMIsStarted() ? PCR_FIRST + DRTM_PCR
                                            : PCR_FIRST + HCRTM_PCR;

    // If there is no DRTM sequence object, then _TPM_Hash_Start
    // was not called so this function returns without doing
    // anything.
    if(g_DRTMHandle == TPM_RH_UNASSIGNED)
        return;

    hashObject = (HASH_OBJECT*)HandleToObject(g_DRTMHandle);
    pAssert(hashObject->attributes.eventSeq);

    // For each of the implemented hash algorithms, update the digest with the
    // data provided.
    for(i = 0; i < HASH_COUNT; i++)
    {
        // make sure that the PCR is implemented for this algorithm
        if(PcrIsAllocated(pcrHandle, hashObject->state.hashState[i].hashAlg))
            // Update sequence object
            CryptDigestUpdate(&hashObject->state.hashState[i], dataSize, data);
    }

    return;
}
```

7.164 /tpm/src/events/_TPM_Hash_End.c

```
#include "Tpm.h"

// This function is called to process a _TPM_Hash_End indication.
LIB_EXPORT void _TPM_Hash_End(void)
{
    UINT32       i;
    TPM2B_DIGEST digest;
    HASH_OBJECT* hashObject;
    TPMI_DH_PCR  pcrHandle;

    // If the DRTM handle is not being used, then either _TPM_Hash_Start has not
    // been called, _TPM_Hash_End was previously called, or some other command
    // was executed and the sequence was aborted.
    if(g_DRTMHandle == TPM_RH_UNASSIGNED)
        return;

    // Get DRTM sequence object
    hashObject = (HASH_OBJECT*)HandleToObject(g_DRTMHandle);

    // Is this _TPM_Hash_End after Startup or before
    if(TPMIsStarted())
    {
        // After

        // Reset the DRTM PCR
        PCRResetDynamics();

        // Extend the DRTM_PCR.
        pcrHandle = PCR_FIRST + DRTM_PCR;

        // DRTM sequence increments restartCount
        gr.restartCount++;
    }
    else
    {
        pcrHandle        = PCR_FIRST + HCRTM_PCR;
        g_DrtmPreStartup = TRUE;
    }

    // Complete hash and extend PCR, or if this is an HCRTM, complete
    // the hash, reset the H-CRTM register (PCR[0]) to 0...04, and then
    // extend the H-CRTM data
    for(i = 0; i < HASH_COUNT; i++)
    {
        TPMI_ALG_HASH hash = CryptHashGetAlgByIndex(i);
        // make sure that the PCR is implemented for this algorithm
        if(PcrIsAllocated(pcrHandle, hashObject->state.hashState[i].hashAlg))
        {
            // Complete hash
            digest.t.size = CryptHashGetDigestSize(hash);
            CryptHashEnd2B(&hashObject->state.hashState[i], &digest.b);

            PcrDrtm(pcrHandle, hash, &digest);
        }
    }

    // Flush sequence object.
    FlushObject(g_DRTMHandle);

    g_DRTMHandle = TPM_RH_UNASSIGNED;

    return;
}
```

7.165 /tpm/src/events/_TPM_Hash_Start.c

```
#include "Tpm.h"

// This function is called to process a _TPM_Hash_Start indication.
LIB_EXPORT void _TPM_Hash_Start(void)
{
    TPM_RC         result;
    TPMI_DH_OBJECT handle;

    // If a DRTM sequence object exists, free it up
    if(g_DRTMHandle != TPM_RH_UNASSIGNED)
    {
        FlushObject(g_DRTMHandle);
        g_DRTMHandle = TPM_RH_UNASSIGNED;
    }

    // Create an event sequence object and store the handle in global
    // g_DRTMHandle. A TPM_RC_OBJECT_MEMORY error may be returned at this point
    // The NULL value for the first parameter will cause the sequence structure to
    // be allocated without being set as present. This keeps the sequence from
    // being left behind if the sequence is terminated early.
    result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);

    // If a free slot was not available, then free up a slot.
    if(result != TPM_RC_SUCCESS)
    {
        // An implementation does not need to have a fixed relationship between
        // slot numbers and handle numbers. To handle the general case, scan for
        // a handle that is assigned and free it for the DRTM sequence.
        // In the reference implementation, the relationship between handles and
        // slots is fixed. So, if the call to ObjectCreateEvenSequence()
        // failed indicating that all slots are occupied, then the first handle we
        // are going to check (TRANSIENT_FIRST) will be occupied. It will be freed
        // so that it can be assigned for use as the DRTM sequence object.
        for(handle = TRANSIENT_FIRST; handle < TRANSIENT_LAST; handle++)
        {
            // try to flush the first object
            if(IsObjectPresent(handle))
                break;
        }
        // If the first call to find a slot fails but none of the slots is occupied
        // then there's a big problem
        pAssert(handle < TRANSIENT_LAST);

        // Free the slot
        FlushObject(handle);

        // Try to create an event sequence object again.  This time, we must
        // succeed.
        result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);
        if(result != TPM_RC_SUCCESS)
            FAIL(FATAL_ERROR_INTERNAL);
    }

    return;
}
```

7.166 /tpm/src/events/_TPM_Init.c

```
#include <private/Tpm.h>
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/prototypes/_TPM_Init_fp.h>
```

```
// This function is used to process a _TPM_Init indication.
LIB_EXPORT void _TPM_Init(void)
{
    g_powerWasLost = g_powerWasLost | _plat__WasPowerLost();

#if SIMULATION && DEBUG
    // If power was lost and this was a simulation, put canary in RAM used by NV
    // so that uninitialized memory can be detected more easily
    if(g_powerWasLost)
    {
        memset(&gc, 0xbb, sizeof(gc));
        memset(&gr, 0xbb, sizeof(gr));
        memset(&gp, 0xbb, sizeof(gp));
        memset(&go, 0xbb, sizeof(go));
    }
#endif

#if ALLOW_FORCE_FAILURE_MODE
    // Clear the flag that forces failure on self-test
    g_forceFailureMode = FALSE;
#endif

    // Disable the tick processing
#if ACT_SUPPORT
    _plat__ACT_EnableTicks(FALSE);
#endif

    // Set initialization state
    TPMInit();

    // Set g_DRTMHandle as unassigned
    g_DRTMHandle = TPM_RH_UNASSIGNED;

    // No H-CRTM, yet.
    g_DrtmPreStartup = FALSE;

    // Initialize the NvEnvironment.
    g_nvOk = NvPowerOn();

    // Initialize cryptographic functions
    g_inFailureMode = (g_nvOk == FALSE) || (CryptInit() == FALSE);
    if(!g_inFailureMode)
    {
        // Load the persistent data
        NvReadPersistent();

        // Load the orderly data (clock and DRBG state).
        // If this is not done here, things break
        NvRead(&go, NV_ORDERLY_DATA, sizeof(go));

        // Start clock. Need to do this after NV has been restored.
        TimePowerOn();
    }
    return;
}
```

7.167 /tpm/src/main/CommandDispatcher.c

```
//* Includes and Typedefs
#include "Tpm.h"
#include "Marshal.h"

#if TABLE_DRIVEN_DISPATCH || TABLE_DRIVEN_MARSHAL

typedef TPM_RC(NoFlagFunction)(void* target, BYTE** buffer, INT32* size);
```

```
typedef TPM_RC(FlagFunction)(void* target, BYTE** buffer, INT32* size, BOOL flag);

typedef FlagFunction* UNMARSHAL_t;

typedef INT16(MarshalFunction)(void* source, BYTE** buffer, INT32* size);
typedef MarshalFunction* MARSHAL_t;

typedef TPM_RC(COMMAND_NO_ARGS)(void);
typedef TPM_RC(COMMAND_IN_ARG)(void* in);
typedef TPM_RC(COMMAND_OUT_ARG)(void* out);
typedef TPM_RC(COMMAND_INOUT_ARG)(void* in, void* out);

typedef union COMMAND_t
{
    COMMAND_NO_ARGS*   noArgs;
    COMMAND_IN_ARG*    inArg;
    COMMAND_OUT_ARG*   outArg;
    COMMAND_INOUT_ARG* inOutArg;
} COMMAND_t;

// This structure is used by ParseHandleBuffer() and CommandDispatcher(). The
// parameters in this structure are unique for each command. The parameters are:
// command      holds the address of the command processing function that is called
//              by Command Dispatcher
// inSize       This is the size of the command-dependent input structure. The
//              input structure holds the unmarshaled handles and command
//              parameters. If the command takes no arguments (handles or
//              parameters) then inSize will have a value of 0.
// outSize      This is the size of the command-dependent output structure. The
//              output structure holds the results of the command in an unmarshaled
//              form. When command processing is completed, these values are
//              marshaled into the output buffer. It is always the case that the
//              unmarshaled version of an output structure is larger then the
//              marshaled version. This is because the marshaled version contains
//              the exact same number of significant bytes but with padding removed.
// typesOffsets    This parameter points to the list of data types that are to be
//              marshaled or unmarshaled. The list of types follows the 'offsets'
//              array. The offsets array is variable sized so the typesOffset filed
//              is necessary for the handle and command processing to be able to
//              find the types that are being handled. The 'offsets' array may be
//              empty. The 'types' structure is described below.
// offsets      This is an array of offsets of each of the parameters in the
//              command or response. When processing the command parameters (not
//              handles) the list contains the offset of the next parameter. For
//              example, if the first command parameter has a size of 4 and there is
//              a second command parameter, then the offset would be 4, indicating
//              that the second parameter starts at 4. If the second parameter has
//              a size of 8, and there is a third parameter, then the second entry
//              in offsets is 12 (4 for the first parameter and 8 for the second).
//              An offset value of 0 in the list indicates the start of the response
//              parameter list. When CommandDispatcher hits this value, it will stop
//              unmarshaling the parameters and call 'command'. If a command has no
//              response parameters and only one command parameter, then offsets can
//              be an empty list.

typedef struct COMMAND_DESCRIPTOR_t
{
    COMMAND_t command;      // Address of the command
    UINT16    inSize;       // Maximum size of the input structure
    UINT16    outSize;      // Maximum size of the output structure
    UINT16    typesOffset;  // address of the types field
    UINT16    offsets[1];
} COMMAND_DESCRIPTOR_t;

// The 'types' list is an encoded byte array. The byte value has two parts. The most
// significant bit is used when a parameter takes a flag and indicates if the flag
```

```
// should be SET or not. The remaining 7 bits are an index into an array of
// addresses of marshaling and unmarshaling functions.
// The array of functions is divided into 6 sections with a value assigned
// to denote the start of that section (and the end of the previous section). The
// defined offset values for each section are:
// 0                               unmarshaling for handles that do not take flags
// HANDLE_FIRST_FLAG_TYPE          unmarshaling for handles that take flags
// PARAMETER_FIRST_TYPE            unmarshaling for parameters that do not take flags
// PARAMETER_FIRST_FLAG_TYPE       unmarshaling for parameters that take flags
// PARAMETER_LAST_TYPE + 1         marshaling for handles
// RESPONSE_PARAMETER_FIRST_TYPE   marshaling for parameters
// RESPONSE_PARAMETER_LAST_TYPE    is the last value in the list of marshaling and
//                                 unmarshaling functions.
//
// The types list is constructed with a byte of 0xff at the end of the command
// parameters and with an 0xff at the end of the response parameters.

#  if COMPRESSED_LISTS
#    define PAD_LIST 0
#  else
#    define PAD_LIST 1
#  endif
#  define _COMMAND_TABLE_DISPATCH_
#  include "CommandDispatchData.h"

#  define TEST_COMMAND TPM_CC_Startup

#  define NEW_CC

#else

#  include "Commands.h"

#endif

//* Marshal/Unmarshal Functions

//** ParseHandleBuffer()
// This is the table-driven version of the handle buffer unmarshaling code
TPM_RC
ParseHandleBuffer(COMMAND* command)
{
    TPM_RC result;
#if TABLE_DRIVEN_DISPATCH || TABLE_DRIVEN_MARSHAL
    COMMAND_DESCRIPTOR_t* desc;
    BYTE*                 types;
    BYTE                  type;
    BYTE                  dType;

    // Make sure that nothing strange has happened
    pAssert(
        command->index < sizeof(s_CommandDataArray) / sizeof(COMMAND_DESCRIPTOR_t*));
    // Get the address of the descriptor for this command
    desc = s_CommandDataArray[command->index];

    pAssert(desc != NULL);
    // Get the associated list of unmarshaling data types.
    types = &((BYTE*)desc)[desc->typesOffset];

    //    if(s_ccAttr[commandIndex].commandIndex == TEST_COMMAND)
    //        commandIndex = commandIndex;
    // No handles yet
    command->handleNum = 0;

    // Get the first type value
    for(type = *types++;
```

```
        // check each byte to make sure that we have not hit the start
        // of the parameters
        (dType = (type & 0x7F)) < PARAMETER_FIRST_TYPE;
        // get the next type
        type = *types++)
    {
#   if TABLE_DRIVEN_MARSHAL
        marshalIndex_t index;
        index  = unmarshalArray[dType] | ((type & 0x80) ? NULL_FLAG : 0);
        result = Unmarshal(index,
                           &(command->handles[command->handleNum]),
                           &command->parameterBuffer,
                           &command->parameterSize);

#   else
        // See if unmarshaling of this handle type requires a flag
        if(dType < HANDLE_FIRST_FLAG_TYPE)
        {
            // Look up the function to do the unmarshaling
            NoFlagFunction* f = (NoFlagFunction*)unmarshalArray[dType];
            // call it
            result = f(&(command->handles[command->handleNum]),
                       &command->parameterBuffer,
                       &command->parameterSize);
        }
        else
        {
            //  Look up the function
            FlagFunction* f = unmarshalArray[dType];

            // Call it setting the flag to the appropriate value
            result = f(&(command->handles[command->handleNum]),
                       &command->parameterBuffer,
                       &command->parameterSize,
                       (type & 0x80) != 0);
        }
#   endif

        // Got a handle
        // We do this first so that the match for the handle offset of the
        // response code works correctly.
        command->handleNum += 1;
        if(result != TPM_RC_SUCCESS)
            // if the unmarshaling failed, return the response code with the
            // handle indication set
            return result + TPM_RC_H + (command->handleNum * TPM_RC_1);
    }
#else
    BYTE**      handleBufferStart   = &command->parameterBuffer;
    INT32*      bufferRemainingSize = &command->parameterSize;
    TPM_HANDLE* handles             = &command->handles[0];
    UINT32*     handleCount         = &command->handleNum;
    *handleCount                    = 0;
    switch(command->code)
    {
#   include "HandleProcess.h"
#   undef handles
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
#endif
    return TPM_RC_SUCCESS;
}

//** CommandDispatcher()
```

```
// Function to unmarshal the command parameters, call the selected action code, and
// marshal the response parameters.
TPM_RC
CommandDispatcher(COMMAND* command)
{
#if !TABLE_DRIVEN_DISPATCH || TABLE_DRIVEN_MARSHAL
    TPM_RC      result;
    BYTE**      paramBuffer     = &command->parameterBuffer;
    INT32*      paramBufferSize = &command->parameterSize;
    BYTE**      responseBuffer  = &command->responseBuffer;
    INT32*      respParmSize    = &command->parameterSize;
    INT32       rSize;
    TPM_HANDLE* handles = &command->handles[0];
    //
    command->handleNum = 0;           // The command-specific code knows how
                                      // many handles there are. This is for
                                      // cataloging the number of response
                                      // handles
    MemoryIoBufferAllocationReset();  // Initialize so that allocation will
                                      // work properly
    switch(GetCommandCode(command->index))
    {
#  include "CommandDispatcher.h"

        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
Exit:
    MemoryIoBufferZero();
    return result;
#else
    COMMAND_DESCRIPTOR_t* desc;
    BYTE*                 types;
    BYTE                  type;
    UINT16*               offsets;
    UINT16                offset = 0;
    UINT32                maxInSize;
    BYTE*                 commandIn;
    INT32                 maxOutSize;
    BYTE*                 commandOut;
    COMMAND_t             cmd;
    TPM_HANDLE*           handles;
    UINT32                hasInParameters  = 0;
    BOOL                  hasOutParameters = FALSE;
    UINT32                pNum             = 0;
    BYTE                  dType;  // dispatch type
    TPM_RC                result;
    //
    // Get the address of the descriptor for this command
    pAssert(
        command->index < sizeof(s_CommandDataArray) / sizeof(COMMAND_DESCRIPTOR_t*));
    desc = s_CommandDataArray[command->index];

    // Get the list of parameter types for this command
    pAssert(desc != NULL);
    types = &((BYTE*)desc)[desc->typesOffset];

    // Get a pointer to the list of parameter offsets
    offsets = &desc->offsets[0];
    // pointer to handles
    handles = command->handles;

    // Get the size required to hold all the unmarshaled parameters for this command
    maxInSize = desc->inSize;
    // and the size of the output parameter structure returned by this command
```

```
    maxOutSize = desc->outSize;

    MemoryIoBufferAllocationReset();
    // Get a buffer for the input parameters
    commandIn = MemoryGetInBuffer(maxInSize);
    // And the output parameters
    commandOut = (BYTE*)MemoryGetOutBuffer((UINT32)maxOutSize);

    // Get the address of the action code dispatch
    cmd = desc->command;

    // Copy any handles into the input buffer
    for(type = *types++; (type & 0x7F) < PARAMETER_FIRST_TYPE; type = *types++)
    {
        // 'offset' was initialized to zero so the first unmarshaling will always
        // be to the start of the data structure
        *(TPM_HANDLE*)&(commandIn[offset]) = *handles++;
        // This check is used so that we don't have to add an additional offset
        // value to the offsets list to correspond to the stop value in the
        // command parameter list.
        if(*types != 0xFF)
            offset = *offsets++;
        //        maxInSize -= sizeof(TPM_HANDLE);
        hasInParameters++;
    }
    // Exit loop with type containing the last value read from types
    // maxInSize has the amount of space remaining in the command action input
    // buffer. Make sure that we don't have more data to unmarshal than is going to
    // fit.

    // type contains the last value read from types so it is not necessary to
    // reload it, which is good because *types now points to the next value
    for(; (dType = (type & 0x7F)) <= PARAMETER_LAST_TYPE; type = *types++)
    {
        pNum++;
#   if TABLE_DRIVEN_MARSHAL
        {
            marshalIndex_t index = unmarshalArray[dType];
            index |= (type & 0x80) ? NULL_FLAG : 0;
            result = Unmarshal(index,
                               &commandIn[offset],
                               &command->parameterBuffer,
                               &command->parameterSize);
        }
#   else
        if(dType < PARAMETER_FIRST_FLAG_TYPE)
        {
            NoFlagFunction* f = (NoFlagFunction*)unmarshalArray[dType];
            result            = f(&commandIn[offset],
                      &command->parameterBuffer,
                      &command->parameterSize);
        }
        else
        {
            FlagFunction* f = unmarshalArray[dType];
            result          = f(&commandIn[offset],
                      &command->parameterBuffer,
                      &command->parameterSize,
                      (type & 0x80) != 0);
        }
#   endif
        if(result != TPM_RC_SUCCESS)
        {
            result += TPM_RC_P + (TPM_RC_1 * pNum);
            goto Exit;
        }
```

```
        // This check is used so that we don't have to add an additional offset
        // value to the offsets list to correspond to the stop value in the
        // command parameter list.
        if(*types != 0xFF)
            offset = *offsets++;
        hasInParameters++;
    }
    // Should have used all the bytes in the input
    if(command->parameterSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }

    // The command parameter unmarshaling stopped when it hit a value that was out
    // of range for unmarshaling values and left *types pointing to the first
    // marshaling type. If that type happens to be the STOP value, then there
    // are no response parameters. So, set the flag to indicate if there are
    // output parameters.
    hasOutParameters = *types != 0xFF;

    // There are four cases for calling, with and without input parameters and with
    // and without output parameters.
    if(hasInParameters > 0)
    {
        if(hasOutParameters)
            result = cmd.inOutArg(commandIn, commandOut);
        else
            result = cmd.inArg(commandIn);
    }
    else
    {
        if(hasOutParameters)
            result = cmd.outArg(commandOut);
        else
            result = cmd.noArgs();
    }
    if(result != TPM_RC_SUCCESS)
        goto Exit;

    // Offset in the marshaled output structure
    offset = 0;

    // Process the return handles, if any
    command->handleNum = 0;

    // Could make this a loop to process output handles but there is only ever
    // one handle in the outputs (for now).
    type = *types++;
    if((dType = (type & 0x7F)) < RESPONSE_PARAMETER_FIRST_TYPE)
    {
        // The out->handle value was referenced as TPM_HANDLE in the
        // action code so it has to be properly aligned.
        command->handles[command->handleNum++] =
            *((TPM_HANDLE*)&(commandOut[offset]));
        maxOutSize -= sizeof(UINT32);
        type   = *types++;
        offset = *offsets++;
    }
    // Use the size of the command action output buffer as the maximum for the
    // number of bytes that can get marshaled. Since the marshaling code has
    // no pointers to data, all of the data being returned has to be in the
    // command action output buffer. If we try to marshal more bytes than
    // could fit into the output buffer, we need to fail.
    for(; (dType = (type & 0x7F)) <= RESPONSE_PARAMETER_LAST_TYPE && !g_inFailureMode;
        type = *types++)
```

```
    {
#   if TABLE_DRIVEN_MARSHAL
        marshalIndex_t index = marshalArray[dType];
        command->parameterSize += Marshal(
            index, &commandOut[offset], &command->responseBuffer, &maxOutSize);
#   else
        const MARSHAL_t f = marshalArray[dType];

        command->parameterSize +=
            f(&commandOut[offset], &command->responseBuffer, &maxOutSize);
#   endif
        offset = *offsets++;
    }
    result = (maxOutSize < 0) ? TPM_RC_FAILURE : TPM_RC_SUCCESS;
Exit:
    MemoryIoBufferZero();
    return result;
#endif
}
```

7.168 /tpm/src/main/ExecCommand.c

```
//** Introduction
//
// This file contains the entry function ExecuteCommand() which provides the main
// control flow for TPM command execution.

//** Includes

#include "Tpm.h"
#include "Marshal.h"
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/prototypes/ExecCommand_fp.h>

// Uncomment this next #include if doing static command/response buffer sizing
// #include "CommandResponseSizes_fp.h"

//** ExecuteCommand()
//
// The function performs the following steps.
//
//  a)  Parses the command header from input buffer.
//  b)  Calls ParseHandleBuffer() to parse the handle area of the command.
//  c)  Validates that each of the handles references a loaded entity.
//  d)  Calls ParseSessionBuffer () to:
//      1)  unmarshal and parse the session area;
//      2)  check the authorizations; and
//      3)  when necessary, decrypt a parameter.
//  e)  Calls CommandDispatcher() to:
//      1)  unmarshal the command parameters from the command buffer;
//      2)  call the routine that performs the command actions; and
//      3)  marshal the responses into the response buffer.
//  f)  If any error occurs in any of the steps above create the error response
//      and return.
//  g)  Calls BuildResponseSession() to:
//      1)  when necessary, encrypt a parameter
//      2)  build the response authorization sessions
//      3)  update the audit sessions and nonces
//  h)  Calls BuildResponseHeader() to complete the construction of the response.
//
// 'responseSize' is set by the caller to the maximum number of bytes available in
// the output buffer. ExecuteCommand will adjust the value and return the number
// of bytes placed in the buffer.
//
```

```
// 'response' is also set by the caller to indicate the buffer into which
//  ExecuteCommand is to place the response.
//
//  'request' and 'response' may point to the same buffer
//
// Note: As of February, 2016, the failure processing has been moved to the
// platform-specific code. When the TPM code encounters an unrecoverable failure, it
// will SET g_inFailureMode and call _plat__Fail(). That function should not return
// but may call ExecuteCommand().
//
LIB_EXPORT void ExecuteCommand(
    uint32_t        requestSize,   // IN: command buffer size
    unsigned char*  request,       // IN: command buffer
    uint32_t*       responseSize,  // IN/OUT: response buffer size
    unsigned char** response       // IN/OUT: response buffer
)
{
    // Command local variables
    UINT32  commandSize;
    COMMAND command;

    // Response local variables
    UINT32 maxResponse = *responseSize;
    TPM_RC result;  // return code for the command

    // This next function call is used in development to size the command and response
    // buffers. The values printed are the sizes of the internal structures and
    // not the sizes of the canonical forms of the command response structures. Also,
    // the sizes do not include the tag, command.code, requestSize, or the
authorization
    // fields.
    //CommandResponseSizes();
    // Set flags for NV access state. This should happen before any other
    // operation that may require a NV write. Note, that this needs to be done
    // even when in failure mode. Otherwise, g_updateNV would stay SET while in
    // Failure mode and the NV would be written on each call.
    g_updateNV     = UT_NONE;
    g_clearOrderly = FALSE;
    if(g_inFailureMode)
    {
        // Do failure mode processing
        TpmFailureMode(requestSize, request, responseSize, response);
        return;
    }
    // Query platform to get the NV state.  The result state is saved internally
    // and will be reported by NvIsAvailable(). The reference code requires that
    // accessibility of NV does not change during the execution of a command.
    // Specifically, if NV is available when the command execution starts and then
    // is not available later when it is necessary to write to NV, then the TPM
    // will go into failure mode.
    NvCheckState();

    // Due to the limitations of the simulation, TPM clock must be explicitly
    // synchronized with the system clock whenever a command is received.
    // This function call is not necessary in a hardware TPM. However, taking
    // a snapshot of the hardware timer at the beginning of the command allows
    // the time value to be consistent for the duration of the command execution.
    TimeUpdateToCurrent();

    // Any command through this function will unceremoniously end the
    // _TPM_Hash_Data/_TPM_Hash_End sequence.
    if(g_DRTMHandle != TPM_RH_UNASSIGNED)
        ObjectTerminateEvent();

    // Get command buffer size and command buffer.
    command.parameterBuffer = request;
```

```
    command.parameterSize    = requestSize;

    // Parse command header: tag, commandSize and command.code.
    // First parse the tag. The unmarshaling routine will validate
    // that it is either TPM_ST_SESSIONS or TPM_ST_NO_SESSIONS.
    result = TPMI_ST_COMMAND_TAG_Unmarshal(
        &command.tag, &command.parameterBuffer, &command.parameterSize);
    if(result != TPM_RC_SUCCESS)
        goto Cleanup;
    // Unmarshal the commandSize indicator.
    result = UINT32_Unmarshal(
        &commandSize, &command.parameterBuffer, &command.parameterSize);
    if(result != TPM_RC_SUCCESS)
        goto Cleanup;
    // On a TPM that receives bytes on a port, the number of bytes that were
    // received on that port is requestSize it must be identical to commandSize.
    // In addition, commandSize must not be larger than MAX_COMMAND_SIZE allowed
    // by the implementation. The check against MAX_COMMAND_SIZE may be redundant
    // as the input processing (the function that receives the command bytes and
    // places them in the input buffer) would likely have the input truncated when
    // it reaches MAX_COMMAND_SIZE, and requestSize would not equal commandSize.
    if(commandSize != requestSize || commandSize > MAX_COMMAND_SIZE)
    {
        result = TPM_RC_COMMAND_SIZE;
        goto Cleanup;
    }
    // Unmarshal the command code.
    result = TPM_CC_Unmarshal(
        &command.code, &command.parameterBuffer, &command.parameterSize);
    if(result != TPM_RC_SUCCESS)
        goto Cleanup;
    // Check to see if the command is implemented.
    command.index = CommandCodeToCommandIndex(command.code);
    if(UNIMPLEMENTED_COMMAND_INDEX == command.index)
    {
        result = TPM_RC_COMMAND_CODE;
        goto Cleanup;
    }
#if FIELD_UPGRADE_IMPLEMENTED == YES
    // If the TPM is in FUM, then the only allowed command is
    // TPM_CC_FieldUpgradeData.
    if(IsFieldUgradeMode() && (command.code != TPM_CC_FieldUpgradeData))
    {
        result = TPM_RC_UPGRADE;
        goto Cleanup;
    }
    else
#endif
        // Excepting FUM, the TPM only accepts TPM2_Startup() after
        // _TPM_Init. After getting a TPM2_Startup(), TPM2_Startup()
        // is no longer allowed.
        if((!TPMIsStarted() && command.code != TPM_CC_Startup)
           || (TPMIsStarted() && command.code == TPM_CC_Startup))
        {
            result = TPM_RC_INITIALIZE;
            goto Cleanup;
        }
    // Start regular command process.
    NvIndexCacheInit();
    // Parse Handle buffer.
    result = ParseHandleBuffer(&command);
    if(result != TPM_RC_SUCCESS)
        goto Cleanup;
    // All handles in the handle area are required to reference TPM-resident
    // entities.
    result = EntityGetLoadStatus(&command);
```

```
    if(result != TPM_RC_SUCCESS)
        goto Cleanup;
    // Authorization session handling for the command.
    ClearCpRpHashes(&command);
    if(command.tag == TPM_ST_SESSIONS)
    {
        // Find out session buffer size.
        result = UINT32_Unmarshal((UINT32*)&command.authSize,
                                  &command.parameterBuffer,
                                  &command.parameterSize);
        if(result != TPM_RC_SUCCESS)
            goto Cleanup;
        // Perform sanity check on the unmarshaled value. If it is smaller than
        // the smallest possible session or larger than the remaining size of
        // the command, then it is an error. NOTE: This check could pass but the
        // session size could still be wrong. That will be determined after the
        // sessions are unmarshaled.
        if(command.authSize < 9 || command.authSize > command.parameterSize)
        {
            result = TPM_RC_SIZE;
            goto Cleanup;
        }
        command.parameterSize -= command.authSize;

        // The actions of ParseSessionBuffer() are described in the introduction.
        // As the sessions are parsed command.parameterBuffer is advanced so, on a
        // successful return, command.parameterBuffer should be pointing at the
        // first byte of the parameters.
        result = ParseSessionBuffer(&command);
        if(result != TPM_RC_SUCCESS)
            goto Cleanup;
    }
    else
    {
        command.authSize = 0;
        // The command has no authorization sessions.
        // If the command requires authorizations, then CheckAuthNoSession() will
        // return an error.
        result = CheckAuthNoSession(&command);
        if(result != TPM_RC_SUCCESS)
            goto Cleanup;
    }
    // Set up the response buffer pointers. CommandDispatch will marshal the
    // response parameters starting at the address in command.responseBuffer.
    //*response = MemoryGetResponseBuffer(command.index);
    // leave space for the command header
    command.responseBuffer = *response + STD_RESPONSE_HEADER;

    // leave space for the parameter size field if needed
    if(command.tag == TPM_ST_SESSIONS)
        command.responseBuffer += sizeof(UINT32);
    if(IsHandleInResponse(command.index))
        command.responseBuffer += sizeof(TPM_HANDLE);

    // CommandDispatcher returns a response handle buffer and a response parameter
    // buffer if it succeeds. It will also set the parameterSize field in the
    // buffer if the tag is TPM_RC_SESSIONS.
    result = CommandDispatcher(&command);
    if(result != TPM_RC_SUCCESS)
        goto Cleanup;

    // Build the session area at the end of the parameter area.
    result = BuildResponseSession(&command);
    if(result != TPM_RC_SUCCESS)
    {
        goto Cleanup;
```

```
    }

Cleanup:
    if(g_clearOrderly == TRUE && NV_IS_ORDERLY)
    {
#if USE_DA_USED
        gp.orderlyState = g_daUsed ? SU_DA_USED_VALUE : SU_NONE_VALUE;
#else
        gp.orderlyState = SU_NONE_VALUE;
#endif
        NV_SYNC_PERSISTENT(orderlyState);
    }
    // This implementation loads an "evict" object to a transient object slot in
    // RAM whenever an "evict" object handle is used in a command so that the
    // access to any object is the same. These temporary objects need to be
    // cleared from RAM whether the command succeeds or fails.
    ObjectCleanupEvict();

    // The parameters and sessions have been marshaled. Now tack on the header and
    // set the sizes
    BuildResponseHeader(&command, *response, result);

    // Try to commit all the writes to NV if any NV write happened during this
    // command execution. This check should be made for both succeeded and failed
    // commands, because a failed one may trigger a NV write in DA logic as well.
    // This is the only place in the command execution path that may call the NV
    // commit. If the NV commit fails, the TPM should be put in failure mode.
    if((g_updateNV != UT_NONE) && !g_inFailureMode)
    {
        if(g_updateNV == UT_ORDERLY)
            NvUpdateIndexOrderlyData();
        if(!NvCommit())
            FAIL(FATAL_ERROR_INTERNAL);
        g_updateNV = UT_NONE;
    }
    pAssert((UINT32)command.parameterSize <= maxResponse);

    // Clear unused bits in response buffer.
    MemorySet(*response + *responseSize, 0, maxResponse - *responseSize);

    // as a final act, and not before, update the response size.
    *responseSize = (UINT32)command.parameterSize;

    return;
}
```

7.169 /tpm/src/main/SessionProcess.c

```
//**  Introduction
// This file contains the subsystem that process the authorization sessions
// including implementation of the Dictionary Attack logic. ExecCommand() uses
// ParseSessionBuffer() to process the authorization session area of a command and
// BuildResponseSession() to create the authorization session area of a response.

//**  Includes and Data Definitions

#define SESSION_PROCESS_C

#include "Tpm.h"
#include "ACT.h"
#include "Marshal.h"

//
//**  Authorization Support Functions
//
```

```
//*** IsDAExempted()
// This function indicates if a handle is exempted from DA logic.
// A handle is exempted if it is:
//  a) a primary seed handle;
//  b) an object with noDA bit SET;
//  c) an NV Index with TPMA_NV_NO_DA bit SET; or
//  d) a PCR handle.
//
//  Return Type: BOOL
//      TRUE(1)         handle is exempted from DA logic
//      FALSE(0)        handle is not exempted from DA logic
BOOL IsDAExempted(TPM_HANDLE handle  // IN: entity handle
)
{
    BOOL result = FALSE;
    //
    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
            // All permanent handles, other than TPM_RH_LOCKOUT, are exempt from
            // DA protection.
            result = (handle != TPM_RH_LOCKOUT);
            break;
        // When this function is called, a persistent object will have been loaded
        // into an object slot and assigned a transient handle.
        case TPM_HT_TRANSIENT:
        {
            TPMA_OBJECT attributes = ObjectGetPublicAttributes(handle);
            result                 = IS_ATTRIBUTE(attributes, TPMA_OBJECT, noDA);
            break;
        }
        case TPM_HT_NV_INDEX:
        {
            NV_INDEX* nvIndex = NvGetIndexInfo(handle, NULL);
            result = IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, NO_DA);
            break;
        }
        case TPM_HT_PCR:
            // PCRs are always exempted from DA.
            result = TRUE;
            break;
        default:
            break;
    }
    return result;
}

//*** IncrementLockout()
// This function is called after an authorization failure that involves use of
// an authValue. If the entity referenced by the handle is not exempt from DA
// protection, then the failedTries counter will be incremented.
//
//  Return Type: TPM_RC
//      TPM_RC_AUTH_FAIL    authorization failure that caused DA lockout to increment
//      TPM_RC_BAD_AUTH     authorization failure did not cause DA lockout to
//                          increment
static TPM_RC IncrementLockout(UINT32 sessionIndex)
{
    TPM_HANDLE handle        = s_associatedHandles[sessionIndex];
    TPM_HANDLE sessionHandle = s_sessionHandles[sessionIndex];
    SESSION*   session       = NULL;
    //
    // Don't increment lockout unless the handle associated with the session
    // is DA protected or the session is bound to a DA protected entity.
    if(sessionHandle == TPM_RS_PW)
```

```
    {
        if(IsDAExempted(handle))
            return TPM_RC_BAD_AUTH;
    }
    else
    {
        session = SessionGet(sessionHandle);
        // If the session is bound to lockout, then use that as the relevant
        // handle. This means that an authorization failure with a bound session
        // bound to lockoutAuth will take precedence over any other
        // lockout check
        if(session->attributes.isLockoutBound == SET)
            handle = TPM_RH_LOCKOUT;
        if(session->attributes.isDaBound == CLEAR
           && (IsDAExempted(handle) || session->attributes.includeAuth == CLEAR))
            // If the handle was changed to TPM_RH_LOCKOUT, this will not return
            // TPM_RC_BAD_AUTH
            return TPM_RC_BAD_AUTH;
    }
    if(handle == TPM_RH_LOCKOUT)
    {
        pAssert(gp.lockOutAuthEnabled == TRUE);

        // lockout is no longer enabled
        gp.lockOutAuthEnabled = FALSE;

        // For TPM_RH_LOCKOUT, if lockoutRecovery is 0, no need to update NV since
        // the lockout authorization will be reset at startup.
        if(gp.lockoutRecovery != 0)
        {
            if(NV_IS_AVAILABLE)
                // Update NV.
                NV_SYNC_PERSISTENT(lockOutAuthEnabled);
            else
                // No NV access for now. Put the TPM in pending mode.
                s_DAPendingOnNV = TRUE;
        }
    }
    else
    {
        if(gp.recoveryTime != 0)
        {
            gp.failedTries++;
            if(NV_IS_AVAILABLE)
                // Record changes to NV. NvWrite will SET g_updateNV
                NV_SYNC_PERSISTENT(failedTries);
            else
                // No NV access for now.  Put the TPM in pending mode.
                s_DAPendingOnNV = TRUE;
        }
    }
    // Register a DA failure and reset the timers.
    DARegisterFailure(handle);

    return TPM_RC_AUTH_FAIL;
}

//*** IsSessionBindEntity()
// This function indicates if the entity associated with the handle is the entity,
// to which this session is bound. The binding would occur by making the "bind"
// parameter in TPM2_StartAuthSession() not equal to TPM_RH_NULL. The binding only
// occurs if the session is an HMAC session. The bind value is a combination of
// the Name and the authValue of the entity.
//
//  Return Type: BOOL
//      TRUE(1)         handle points to the session start entity
```

```
//      FALSE(0)        handle does not point to the session start entity
static BOOL IsSessionBindEntity(
    TPM_HANDLE associatedHandle,  // IN: handle to be authorized
    SESSION*   session            // IN: associated session
)
{
    TPM2B_NAME entity;  // The bind value for the entity
                        //
    // If the session is not bound, return FALSE.
    if(session->attributes.isBound)
    {
        // Compute the bind value for the entity.
        SessionComputeBoundEntity(associatedHandle, &entity);

        // Compare to the bind value in the session.
        return MemoryEqual2B(&entity.b, &session->u1.boundEntity.b);
    }
    return FALSE;
}

//*** IsPolicySessionRequired()
// Checks if a policy session is required for a command. If a command requires
// DUP or ADMIN role authorization, then the handle that requires that role is the
// first handle in the command. This simplifies this checking. If a new command
// is created that requires multiple ADMIN role authorizations, then it will
// have to be special-cased in this function.
// A policy session is required if:
// a) the command requires the DUP role;
// b) the command requires the ADMIN role and the authorized entity
//    is an object and its adminWithPolicy bit is SET;
// c) the command requires the ADMIN role and the authorized entity
//    is a permanent handle or an NV Index; or
// d) the authorized entity is a PCR belonging to a policy group, and
//    has its policy initialized
//  Return Type: BOOL
//      TRUE(1)         policy session is required
//      FALSE(0)        policy session is not required
static BOOL IsPolicySessionRequired(COMMAND_INDEX commandIndex,  // IN: command index
                                    UINT32        sessionIndex   // IN: session index
)
{
    AUTH_ROLE role = CommandAuthRole(commandIndex, sessionIndex);
    TPM_HT    type = HandleGetType(s_associatedHandles[sessionIndex]);
    //
    if(role == AUTH_DUP)
        return TRUE;
    if(role == AUTH_ADMIN)
    {
        // We allow an exception for ADMIN role in a transient object. If the object
        // allows ADMIN role actions with authorization, then policy is not
        // required. For all other cases, there is no way to override the command
        // requirement that a policy be used
        if(type == TPM_HT_TRANSIENT)
        {
            OBJECT* object = HandleToObject(s_associatedHandles[sessionIndex]);

            if(!IS_ATTRIBUTE(
                   object->publicArea.objectAttributes, TPMA_OBJECT, adminWithPolicy))
                return FALSE;
        }
        return TRUE;
    }

    if(type == TPM_HT_PCR)
    {
        if(PCRPolicyIsAvailable(s_associatedHandles[sessionIndex]))
```

```
        {
            TPM2B_DIGEST  policy;
            TPMI_ALG_HASH policyAlg;
            policyAlg = PCRGetAuthPolicy(s_associatedHandles[sessionIndex], &policy);
            if(policyAlg != TPM_ALG_NULL)
                return TRUE;
        }
    }
    return FALSE;
}

//*** IsAuthValueAvailable()
// This function indicates if authValue is available and allowed for USER role
// authorization of an entity.
//
// This function is similar to IsAuthPolicyAvailable() except that it does not
// check the size of the authValue as IsAuthPolicyAvailable() does (a null
// authValue is a valid authorization, but a null policy is not a valid policy).
//
// This function does not check that the handle reference is valid or if the entity
// is in an enabled hierarchy. Those checks are assumed to have been performed
// during the handle unmarshaling.
//
//  Return Type: BOOL
//      TRUE(1)         authValue is available
//      FALSE(0)        authValue is not available
static BOOL IsAuthValueAvailable(TPM_HANDLE    handle,        // IN: handle of entity
                                 COMMAND_INDEX commandIndex,  // IN: command index
                                 UINT32        sessionIndex   // IN: session index
)
{
    BOOL result = FALSE;
    //
    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
            switch(handle)
            {
                    // At this point hierarchy availability has already been
                    // checked so primary seed handles are always available here
                case TPM_RH_OWNER:
                case TPM_RH_ENDORSEMENT:
                case TPM_RH_PLATFORM:
#if VENDOR_PERMANENT_AUTH_ENABLED == YES
                    // This vendor defined handle associated with the
                    // manufacturer's shared secret
                case VENDOR_PERMANENT_AUTH_HANDLE:
#endif
                    // The DA checking has been performed on LockoutAuth but we
                    // bypass the DA logic if we are using lockout policy. The
                    // policy would allow execution to continue an lockoutAuth
                    // could be used, even if direct use of lockoutAuth is disabled
                case TPM_RH_LOCKOUT:
                    // NullAuth is always available.
                case TPM_RH_NULL:
                    result = TRUE;
                    break;

#if ACT_SUPPORT
                    FOR_EACH_ACT(CASE_ACT_HANDLE)
                    {
                        // The ACT auth value is not available if the platform is
disabled
                        result = g_phEnable == SET;
                        break;
                    }
```

```
#endif  // ACT_SUPPORT

                default:
                    // Otherwise authValue is not available.
                    break;
            }
            break;
        case TPM_HT_TRANSIENT:
            // A persistent object has already been loaded and the internal
            // handle changed.
            {
                OBJECT*     object;
                TPMA_OBJECT attributes;
                //
                object     = HandleToObject(handle);
                attributes = object->publicArea.objectAttributes;

                // authValue is always available for a sequence object.
                // An alternative for this is to
                // SET_ATTRIBUTE(object->publicArea, TPMA_OBJECT, userWithAuth) when
the
                // sequence is started.
                if(ObjectIsSequence(object))
                {
                    result = TRUE;
                    break;
                }
                // authValue is available for an object if it has its sensitive
                // portion loaded and
                //  a) userWithAuth bit is SET, or
                //  b) ADMIN role is required
                if(object->attributes.publicOnly == CLEAR
                   && (IS_ATTRIBUTE(attributes, TPMA_OBJECT, userWithAuth)
                       || (CommandAuthRole(commandIndex, sessionIndex) == AUTH_ADMIN
                           && !IS_ATTRIBUTE(
                               attributes, TPMA_OBJECT, adminWithPolicy))))
                    result = TRUE;
            }
            break;
        case TPM_HT_NV_INDEX:
            // NV Index.
            {
                NV_REF    locator;
                NV_INDEX* nvIndex = NvGetIndexInfo(handle, &locator);
                TPMA_NV   nvAttributes;
                //
                pAssert(nvIndex != 0);

                nvAttributes = nvIndex->publicArea.attributes;

                if(IsWriteOperation(commandIndex))
                {
                    // AuthWrite can't be set for a PIN index
                    if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, AUTHWRITE))
                        result = TRUE;
                }
                else
                {
                    // A "read" operation
                    // For a PIN Index, the authValue is available as long as the
                    // Index has been written and the pinCount is less than pinLimit
                    if(IsNvPinFailIndex(nvAttributes)
                       || IsNvPinPassIndex(nvAttributes))
                    {
                        NV_PIN pin;
                        if(!IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITTEN))
```

```
                        break;  // return false
                    // get the index values
                    pin.intVal = NvGetUINT64Data(nvIndex, locator);
                    if(pin.pin.pinCount < pin.pin.pinLimit)
                        result = TRUE;
                }
                // For non-PIN Indexes, need to allow use of the authValue
                else if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, AUTHREAD))
                    result = TRUE;
            }
        }
        break;
    case TPM_HT_PCR:
        // PCR handle.
        // authValue is always allowed for PCR
        result = TRUE;
        break;
    default:
        // Otherwise, authValue is not available
        break;
  }
  return result;
}

//*** IsAuthPolicyAvailable()
// This function indicates if an authPolicy is available and allowed.
//
// This function does not check that the handle reference is valid or if the entity
// is in an enabled hierarchy. Those checks are assumed to have been performed
// during the handle unmarshaling.
//
//  Return Type: BOOL
//      TRUE(1)         authPolicy is available
//      FALSE(0)        authPolicy is not available
static BOOL IsAuthPolicyAvailable(TPM_HANDLE    handle,         // IN: handle of entity
                                  COMMAND_INDEX commandIndex,   // IN: command index
                                  UINT32        sessionIndex    // IN: session index
)
{
  BOOL result = FALSE;
  //
  switch(HandleGetType(handle))
  {
    case TPM_HT_PERMANENT:
        switch(handle)
        {
            // At this point hierarchy availability has already been checked.
            case TPM_RH_OWNER:
                if(gp.ownerPolicy.t.size != 0)
                    result = TRUE;
                break;
            case TPM_RH_ENDORSEMENT:
                if(gp.endorsementPolicy.t.size != 0)
                    result = TRUE;
                break;
            case TPM_RH_PLATFORM:
                if(gc.platformPolicy.t.size != 0)
                    result = TRUE;
                break;
#if ACT_SUPPORT

#  define ACT_GET_POLICY(N)                      \
      case TPM_RH_ACT_##N:                       \
          if(go.ACT_##N.authPolicy.t.size != 0) \
              result = TRUE;                     \
          break;
```

```
                    FOR_EACH_ACT(ACT_GET_POLICY)
#endif  // ACT_SUPPORT

                case TPM_RH_LOCKOUT:
                    if(gp.lockoutPolicy.t.size != 0)
                        result = TRUE;
                    break;
                default:
                    break;
            }
            break;
        case TPM_HT_TRANSIENT:
        {
            // Object handle.
            // An evict object would already have been loaded and given a
            // transient object handle by this point.
            OBJECT* object = HandleToObject(handle);
            // Policy authorization is not available for an object with only
            // public portion loaded.
            if(object->attributes.publicOnly == CLEAR)
            {
                // Policy authorization is always available for an object but
                // is never available for a sequence.
                if(!ObjectIsSequence(object))
                    result = TRUE;
            }
            break;
        }
        case TPM_HT_NV_INDEX:
            // An NV Index.
            {
                NV_INDEX* nvIndex      = NvGetIndexInfo(handle, NULL);
                TPMA_NV   nvAttributes = nvIndex->publicArea.attributes;
                //
                // If the policy size is not zero, check if policy can be used.
                if(nvIndex->publicArea.authPolicy.t.size != 0)
                {
                    // If policy session is required for this handle, always
                    // uses policy regardless of the attributes bit setting
                    if(IsPolicySessionRequired(commandIndex, sessionIndex))
                        result = TRUE;
                    // Otherwise, the presence of the policy depends on the NV
                    // attributes.
                    else if(IsWriteOperation(commandIndex))
                    {
                        if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, POLICYWRITE))
                            result = TRUE;
                    }
                    else
                    {
                        if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, POLICYREAD))
                            result = TRUE;
                    }
                }
            }
            break;
        case TPM_HT_PCR:
            // PCR handle.
            if(PCRPolicyIsAvailable(handle))
                result = TRUE;
            break;
        default:
            break;
    }
    return result;
```

```
}

//**  Session Parsing Functions

//*** ClearCpRpHashes()
void ClearCpRpHashes(COMMAND* command)
{
    // The macros expand according to the implemented hash algorithms. An IDE may
    // complain that COMMAND does not contain SHA1CpHash or SHA1RpHash because of the
    // complexity of the macro expansion where the data space is defined; but, if SHA1
    // is implemented, it actually does  and the compiler is happy.
#define CLEAR_CP_HASH(HASH, Hash) command->Hash##CpHash.b.size = 0;
    FOR_EACH_HASH(CLEAR_CP_HASH)
#define CLEAR_RP_HASH(HASH, Hash) command->Hash##RpHash.b.size = 0;
    FOR_EACH_HASH(CLEAR_RP_HASH)
}

//*** GetCpHashPointer()
// Function to get a pointer to the cpHash of the command
static TPM2B_DIGEST* GetCpHashPointer(COMMAND* command, TPMI_ALG_HASH hashAlg)
{
    TPM2B_DIGEST* retVal;
//
// Define the macro that will expand for each implemented algorithm in the switch
// statement below.
#define GET_CP_HASH_POINTER(HASH, Hash)                 \
    case ALG_##HASH##_VALUE:                            \
        retVal = (TPM2B_DIGEST*)&command->Hash##CpHash; \
        break;

    switch(hashAlg)
    {
        // For each implemented hash, this will expand as defined above
        // by GET_CP_HASH_POINTER. Your IDE may complain that
        // 'struct "COMMAND" has no field "SHA1CpHash"' but the compiler says
        // it does, so...
        FOR_EACH_HASH(GET_CP_HASH_POINTER)
        default:
            retVal = NULL;
            break;
    }
    return retVal;
}

//*** GetRpHashPointer()
// Function to get a pointer to the RpHash of the command
static TPM2B_DIGEST* GetRpHashPointer(COMMAND* command, TPMI_ALG_HASH hashAlg)
{
    TPM2B_DIGEST* retVal;
//
// Define the macro that will expand for each implemented algorithm in the switch
// statement below.
#define GET_RP_HASH_POINTER(HASH, Hash)                 \
    case ALG_##HASH##_VALUE:                            \
        retVal = (TPM2B_DIGEST*)&command->Hash##RpHash; \
        break;

    switch(hashAlg)
    {
        // For each implemented hash, this will expand as defined above
        // by GET_RP_HASH_POINTER. Your IDE may complain that
        // 'struct "COMMAND" has no field 'SHA1RpHash'" but the compiler says
        // it does, so...
        FOR_EACH_HASH(GET_RP_HASH_POINTER)
        default:
            retVal = NULL;
```

```
            break;
    }
    return retVal;
}

//*** ComputeCpHash()
// This function computes the cpHash as defined in Part 2 and described in Part 1.
static TPM2B_DIGEST* ComputeCpHash(COMMAND* command,  // IN: command parsing structure
                                   TPMI_ALG_HASH hashAlg  // IN: hash algorithm
)
{
    UINT32        i;
    HASH_STATE    hashState;
    TPM2B_NAME    name;
    TPM2B_DIGEST* cpHash;
    //
    // cpHash = hash(commandCode [ || authName1
    //                            [ || authName2
    //                            [ || authName 3 ]]]
    //                            [ || parameters])
    // A cpHash can contain just a commandCode only if the lone session is
    // an audit session.
    // Get pointer to the hash value
    cpHash = GetCpHashPointer(command, hashAlg);
    if(cpHash->t.size == 0)
    {
        cpHash->t.size = CryptHashStart(&hashState, hashAlg);
        //  Add commandCode.
        CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), command->code);
        //  Add authNames for each of the handles.
        for(i = 0; i < command->handleNum; i++)
            CryptDigestUpdate2B(&hashState,
                                &EntityGetName(command->handles[i], &name)->b);
        //  Add the parameters.
        CryptDigestUpdate(
            &hashState, command->parameterSize, command->parameterBuffer);
        //  Complete the hash.
        CryptHashEnd2B(&hashState, &cpHash->b);
    }
    return cpHash;
}

//*** GetCpHash()
// This function is used to access a precomputed cpHash.
static TPM2B_DIGEST* GetCpHash(COMMAND* command, TPMI_ALG_HASH hashAlg)
{
    TPM2B_DIGEST* cpHash = GetCpHashPointer(command, hashAlg);
    //
    pAssert(cpHash->t.size != 0);
    return cpHash;
}

//*** CompareTemplateHash()
// This function computes the template hash and compares it to the session
// templateHash. It is the hash of the second parameter
// assuming that the command is TPM2_Create(), TPM2_CreatePrimary(), or
// TPM2_CreateLoaded()
//  Return Type: BOOL
//      TRUE(1)         template hash equal to session->templateHash
//      FALSE(0)        template hash not equal to session->templateHash
static BOOL CompareTemplateHash(COMMAND* command,  // IN: parsing structure
                                SESSION* session   // IN: session data
)
{
    BYTE*        pBuffer = command->parameterBuffer;
    INT32        pSize   = command->parameterSize;
```

```
    TPM2B_DIGEST tHash;
    UINT16       size;
    //
    // Only try this for the three commands for which it is intended
    if(command->code != TPM_CC_Create && command->code != TPM_CC_CreatePrimary
#if CC_CreateLoaded
       && command->code != TPM_CC_CreateLoaded
#endif
    )
        return FALSE;
    // Assume that the first parameter is a TPM2B and unmarshal the size field
    // Note: this will not affect the parameter buffer and size in the calling
    // function.
    if(UINT16_Unmarshal(&size, &pBuffer, &pSize) != TPM_RC_SUCCESS)
        return FALSE;
    // reduce the space in the buffer.
    // NOTE: this could make pSize go negative if the parameters are not correct but
    // the unmarshaling code does not try to unmarshal if the remaining size is
    // negative.
    pSize -= size;

    // Advance the pointer
    pBuffer += size;

    // Get the size of what should be the template
    if(UINT16_Unmarshal(&size, &pBuffer, &pSize) != TPM_RC_SUCCESS)
        return FALSE;
    // See if this is reasonable
    if(size > pSize)
        return FALSE;
    // Hash the template data
    tHash.t.size = CryptHashBlock(
        session->authHashAlg, size, pBuffer, sizeof(tHash.t.buffer), tHash.t.buffer);
    return (MemoryEqual2B(&session->u1.templateHash.b, &tHash.b));
}

//*** CompareNameHash()
// This function computes the name hash and compares it to the nameHash in the
// session data, returning true if they are equal.
BOOL CompareNameHash(COMMAND* command,  // IN: main parsing structure
                     SESSION* session   // IN: session structure with nameHash
)
{
    HASH_STATE   hashState;
    TPM2B_DIGEST nameHash;
    UINT32       i;
    TPM2B_NAME   name;
    //
    nameHash.t.size = CryptHashStart(&hashState, session->authHashAlg);
    //  Add names.
    for(i = 0; i < command->handleNum; i++)
        CryptDigestUpdate2B(&hashState,
                            &EntityGetName(command->handles[i], &name)->b);
    //  Complete hash.
    CryptHashEnd2B(&hashState, &nameHash.b);
    // and compare
    return MemoryEqual(
        session->u1.nameHash.t.buffer, nameHash.t.buffer, nameHash.t.size);
}

//*** CompareParametersHash()
// This function computes the parameters hash and compares it to the pHash in
// the session data, returning true if they are equal.
BOOL CompareParametersHash(COMMAND* command,  // IN: main parsing structure
                           SESSION* session   // IN: session structure with pHash
)
```

```
{
    HASH_STATE   hashState;
    TPM2B_DIGEST pHash;
    //
    pHash.t.size = CryptHashStart(&hashState, session->authHashAlg);
    //  Add commandCode.
    CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), command->code);
    //  Add the parameters.
    CryptDigestUpdate(&hashState, command->parameterSize, command->parameterBuffer);
    //  Complete hash.
    CryptHashEnd2B(&hashState, &pHash.b);
    // and compare
    return MemoryEqual2B(&session->u1.pHash.b, &pHash.b);
}

//*** CheckPWAuthSession()
// This function validates the authorization provided in a PWAP session. It
// compares the input value to authValue of the authorized entity. Argument
// sessionIndex is used to get handles handle of the referenced entities from
// s_inputAuthValues[] and s_associatedHandles[].
//
//  Return Type: TPM_RC
//        TPM_RC_AUTH_FAIL          authorization fails and increments DA failure
//                                  count
//        TPM_RC_BAD_AUTH           authorization fails but DA does not apply
//
static TPM_RC CheckPWAuthSession(
    UINT32 sessionIndex  // IN: index of session to be processed
)
{
    TPM2B_AUTH authValue;
    TPM_HANDLE associatedHandle = s_associatedHandles[sessionIndex];
    //
    // Strip trailing zeros from the password.
    MemoryRemoveTrailingZeros(&s_inputAuthValues[sessionIndex]);

    // Get the authValue with trailing zeros removed
    EntityGetAuthValue(associatedHandle, &authValue);

    // Success if the values are identical.
    if(MemoryEqual2B(&s_inputAuthValues[sessionIndex].b, &authValue.b))
    {
        return TPM_RC_SUCCESS;
    }
    else  // if the digests are not identical
    {
        // Invoke DA protection if applicable.
        return IncrementLockout(sessionIndex);
    }
}

//*** ComputeCommandHMAC()
// This function computes the HMAC for an authorization session in a command.
/*(See part 1 specification -- this tag keeps this comment from showing up in
// merged document which is probably good because this comment doesn't look right.
//      The sessionAuth value
//      authHMAC := HMACsHash((sessionKey | authValue),
//                  (pHash | nonceNewer | nonceOlder  | nonceTPMencrypt-only
//                   | nonceTPMaudit   | sessionAttributes))
// Where:
//      HMACsHash()     The HMAC algorithm using the hash algorithm specified
//                      when the session was started.
//
//      sessionKey      A value that is computed in a protocol-dependent way,
//                      using KDFa. When used in an HMAC or KDF, the size field
//                      for this value is not included.
```

```
//
//      authValue       A value that is found in the sensitive area of an entity.
//                      When used in an HMAC or KDF, the size field for this
//                      value is not included.
//
//      pHash           Hash of the command (cpHash) using the session hash.
//                      When using a pHash in an HMAC computation, only the
//                      digest is used.
//
//      nonceNewer      A value that is generated by the entity using the
//                      session. A new nonce is generated on each use of the
//                      session. For a command, this will be nonceCaller.
//                      When used in an HMAC or KDF, the size field is not used.
//
//      nonceOlder      A TPM2B_NONCE that was received the previous time the
//                      session was used. For a command, this is nonceTPM.
//                      When used in an HMAC or KDF, the size field is not used.
//
//      nonceTPMdecrypt     The nonceTPM of the decrypt session is included in
//                          the HMAC, but only in the command.
//
//      nonceTPMencrypt     The nonceTPM of the encrypt session is included in
//                          the HMAC but only in the command.
//
//      sessionAttributes   A byte indicating the attributes associated with the
//                          particular use of the session.
*/
static TPM2B_DIGEST* ComputeCommandHMAC(
    COMMAND*      command,       // IN: primary control structure
    UINT32        sessionIndex,  // IN: index of session to be processed
    TPM2B_DIGEST* hmac           // OUT: authorization HMAC
)
{
    TPM2B_TYPE(KEY, (sizeof(AUTH_VALUE) * 2));
    TPM2B_KEY    key;
    BYTE         marshalBuffer[sizeof(TPMA_SESSION)];
    BYTE*        buffer;
    UINT32       marshalSize;
    HMAC_STATE   hmacState;
    TPM2B_NONCE* nonceDecrypt;
    TPM2B_NONCE* nonceEncrypt;
    SESSION*     session;
    //
    nonceDecrypt = NULL;
    nonceEncrypt = NULL;

    // Determine if extra nonceTPM values are going to be required.
    // If this is the first session (sessionIndex = 0) and it is an authorization
    // session that uses an HMAC, then check if additional session nonces are to be
    // included.
    if(sessionIndex == 0 && s_associatedHandles[sessionIndex] != TPM_RH_UNASSIGNED)
    {
        // If there is a decrypt session and if this is not the decrypt session,
        // then an extra nonce may be needed.
        if(s_decryptSessionIndex != UNDEFINED_INDEX
           && s_decryptSessionIndex != sessionIndex)
        {
            // Will add the nonce for the decrypt session.
            SESSION* decryptSession =
                SessionGet(s_sessionHandles[s_decryptSessionIndex]);
            nonceDecrypt = &decryptSession->nonceTPM;
        }
        // Now repeat for the encrypt session.
        if(s_encryptSessionIndex != UNDEFINED_INDEX
           && s_encryptSessionIndex != sessionIndex
           && s_encryptSessionIndex != s_decryptSessionIndex)
```

```
        {
            // Have to have the nonce for the encrypt session.
            SESSION* encryptSession =
                SessionGet(s_sessionHandles[s_encryptSessionIndex]);
            nonceEncrypt = &encryptSession->nonceTPM;
        }
    }

    // Continue with the HMAC processing.
    session = SessionGet(s_sessionHandles[sessionIndex]);

    // Generate HMAC key.
    MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));

    // Check if the session has an associated handle and if the associated entity
    // is the one to which the session is bound. If not, add the authValue of
    // this entity to the HMAC key.
    // If the session is bound to the object or the session is a policy session
    // with no authValue required, do not include the authValue in the HMAC key.
    // Note: For a policy session, its isBound attribute is CLEARED.
    //
    // Include the entity authValue if it is needed
    if(session->attributes.includeAuth == SET)
    {
        TPM2B_AUTH authValue;
        // Get the entity authValue with trailing zeros removed
        EntityGetAuthValue(s_associatedHandles[sessionIndex], &authValue);
        // add the authValue to the HMAC key
        MemoryConcat2B(&key.b, &authValue.b, sizeof(key.t.buffer));
    }
    // if the HMAC key size is 0, a NULL string HMAC is allowed
    if(key.t.size == 0 && s_inputAuthValues[sessionIndex].t.size == 0)
    {
        hmac->t.size = 0;
        return hmac;
    }
    // Start HMAC
    hmac->t.size = CryptHmacStart2B(&hmacState, session->authHashAlg, &key.b);

    //  Add cpHash
    CryptDigestUpdate2B(&hmacState.hashState,
                        &ComputeCpHash(command, session->authHashAlg)->b);
    //  Add nonces as required
    CryptDigestUpdate2B(&hmacState.hashState, &s_nonceCaller[sessionIndex].b);
    CryptDigestUpdate2B(&hmacState.hashState, &session->nonceTPM.b);
    if(nonceDecrypt != NULL)
        CryptDigestUpdate2B(&hmacState.hashState, &nonceDecrypt->b);
    if(nonceEncrypt != NULL)
        CryptDigestUpdate2B(&hmacState.hashState, &nonceEncrypt->b);
    //  Add sessionAttributes
    buffer      = marshalBuffer;
    marshalSize = TPMA_SESSION_Marshal(&(s_attributes[sessionIndex]), &buffer, NULL);
    CryptDigestUpdate(&hmacState.hashState, marshalSize, marshalBuffer);
    // Complete the HMAC computation
    CryptHmacEnd2B(&hmacState, &hmac->b);

    return hmac;
}

//*** CheckSessionHMAC()
// This function checks the HMAC of in a session. It uses ComputeCommandHMAC()
// to compute the expected HMAC value and then compares the result with the
// HMAC in the authorization session. The authorization is successful if they
// are the same.
//
// If the authorizations are not the same, IncrementLockout() is called. It will
```

```
// return TPM_RC_AUTH_FAIL if the failure caused the failureCount to increment.
// Otherwise, it will return TPM_RC_BAD_AUTH.
//
//  Return Type: TPM_RC
//      TPM_RC_AUTH_FAIL        authorization failure caused failureCount increment
//      TPM_RC_BAD_AUTH         authorization failure did not cause failureCount
//                              increment
//
static TPM_RC CheckSessionHMAC(
    COMMAND* command,      // IN: primary control structure
    UINT32   sessionIndex  // IN: index of session to be processed
)
{
    TPM2B_DIGEST hmac;  // authHMAC for comparing
                        //
    // Compute authHMAC
    ComputeCommandHMAC(command, sessionIndex, &hmac);

    // Compare the input HMAC with the authHMAC computed above.
    if(!MemoryEqual2B(&s_inputAuthValues[sessionIndex].b, &hmac.b))
    {
        // If an HMAC session has a failure, invoke the anti-hammering
        // if it applies to the authorized entity or the session.
        // Otherwise, just indicate that the authorization is bad.
        return IncrementLockout(sessionIndex);
    }
    return TPM_RC_SUCCESS;
}

//*** CheckPolicyAuthSession()
//  This function is used to validate the authorization in a policy session.
//  This function performs the following comparisons to see if a policy
//  authorization is properly provided. The check are:
//  a) compare policyDigest in session with authPolicy associated with
//     the entity to be authorized;
//  b) compare timeout if applicable;
//  c) compare commandCode if applicable;
//  d) compare cpHash if applicable; and
//  e) see if PCR values have changed since computed.
//
// If all the above checks succeed, the handle is authorized.
// The order of these comparisons is not important because any failure will
// result in the same error code.
//
//  Return Type: TPM_RC
//      TPM_RC_PCR_CHANGED          PCR value is not current
//      TPM_RC_POLICY_FAIL          policy session fails
//      TPM_RC_LOCALITY             command locality is not allowed
//      TPM_RC_POLICY_CC            CC doesn't match
//      TPM_RC_EXPIRED              policy session has expired
//      TPM_RC_PP                   PP is required but not asserted
//      TPM_RC_NV_UNAVAILABLE       NV is not available for write
//      TPM_RC_NV_RATE              NV is rate limiting
static TPM_RC CheckPolicyAuthSession(
    COMMAND* command,      // IN: primary parsing structure
    UINT32   sessionIndex  // IN: index of session to be processed
)
{
    SESSION*      session;
    TPM2B_DIGEST  authPolicy;
    TPMI_ALG_HASH policyAlg;
    UINT8         locality;
    //
    // Initialize pointer to the authorization session.
    session = SessionGet(s_sessionHandles[sessionIndex]);
```

```
// If the command is TPM2_PolicySecret(), make sure that
// either password or authValue is required
if(command->code == TPM_CC_PolicySecret
   && session->attributes.isPasswordNeeded == CLEAR
   && session->attributes.isAuthValueNeeded == CLEAR)
    return TPM_RC_MODE;
// See if the PCR counter for the session is still valid.
if(!SessionPCRValueIsCurrent(session))
    return TPM_RC_PCR_CHANGED;
// Get authPolicy.
policyAlg = EntityGetAuthPolicy(s_associatedHandles[sessionIndex], &authPolicy);
// Compare authPolicy.
if(!MemoryEqual2B(&session->u2.policyDigest.b, &authPolicy.b))
    return TPM_RC_POLICY_FAIL;
// Policy is OK so check if the other factors are correct

// Compare policy hash algorithm.
if(policyAlg != session->authHashAlg)
    return TPM_RC_POLICY_FAIL;

// Compare timeout.
if(session->timeout != 0)
{
    // Cannot compare time if clock stop advancing.  An TPM_RC_NV_UNAVAILABLE
    // or TPM_RC_NV_RATE error may be returned here. This doesn't mean that
    // a new nonce will be created just that, because TPM time can't advance
    // we can't do time-based operations.
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    if((session->timeout < g_time) || (session->epoch != g_timeEpoch))
        return TPM_RC_EXPIRED;
}
// If command code is provided it must match
if(session->commandCode != 0)
{
    if(session->commandCode != command->code)
        return TPM_RC_POLICY_CC;
}
else
{
    // If command requires a DUP or ADMIN authorization, the session must have
    // command code set.
    AUTH_ROLE role = CommandAuthRole(command->index, sessionIndex);
    if(role == AUTH_ADMIN || role == AUTH_DUP)
        return TPM_RC_POLICY_FAIL;
}
// Check command locality.
{
    BYTE  sessionLocality[sizeof(TPMA_LOCALITY)];
    BYTE* buffer = sessionLocality;

    // Get existing locality setting in canonical form
    sessionLocality[0] = 0;
    TPMA_LOCALITY_Marshal(&session->commandLocality, &buffer, NULL);

    // See if the locality has been set
    if(sessionLocality[0] != 0)
    {
        // If so, get the current locality
        locality = _plat__LocalityGet();
        if(locality < 5)
        {
            if(((sessionLocality[0] & (1 << locality)) == 0)
               || sessionLocality[0] > 31)
                return TPM_RC_LOCALITY;
        }
```

```
            else if(locality > 31)
            {
                if(sessionLocality[0] != locality)
                    return TPM_RC_LOCALITY;
            }
            else
            {
                // Could throw an assert here but a locality error is just
                // as good. It just means that, whatever the locality is, it isn't
                // the locality requested so...
                return TPM_RC_LOCALITY;
            }
        }
    }  // end of locality check
    // Check physical presence.
    if(session->attributes.isPPRequired == SET && !_plat__PhysicalPresenceAsserted())
        return TPM_RC_PP;
    // Compare cpHash/nameHash/pHash/templateHash if defined.
    if(session->u1.cpHash.b.size != 0)
    {
        BOOL OK = FALSE;
        if(session->attributes.isCpHashDefined)
            // Compare cpHash.
            OK = MemoryEqual2B(&session->u1.cpHash.b,
                               &ComputeCpHash(command, session->authHashAlg)->b);
        else if(session->attributes.isNameHashDefined)
            OK = CompareNameHash(command, session);
        else if(session->attributes.isParametersHashDefined)
            OK = CompareParametersHash(command, session);
        else if(session->attributes.isTemplateHashDefined)
            OK = CompareTemplateHash(command, session);
        if(!OK)
            return TPM_RCS_POLICY_FAIL;
    }
    if(session->attributes.checkNvWritten)
    {
        NV_REF    locator;
        NV_INDEX* nvIndex;
        //
        // If this is not an NV index, the policy makes no sense so fail it.
        if(HandleGetType(s_associatedHandles[sessionIndex]) != TPM_HT_NV_INDEX)
            return TPM_RC_POLICY_FAIL;
        // Get the index data
        nvIndex = NvGetIndexInfo(s_associatedHandles[sessionIndex], &locator);

        // Make sure that the TPMA_WRITTEN_ATTRIBUTE has the desired state
        if((IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
           != (session->attributes.nvWrittenState == SET))
            return TPM_RC_POLICY_FAIL;
    }
    return TPM_RC_SUCCESS;
}

//*** RetrieveSessionData()
// This function will unmarshal the sessions in the session area of a command. The
// values are placed in the arrays that are defined at the beginning of this file.
// The normal unmarshaling errors are possible.
//
//  Return Type: TPM_RC
//      TPM_RC_SUCCSS       unmarshaled without error
//      TPM_RC_SIZE         the number of bytes unmarshaled is not the same
//                          as the value for authorizationSize in the command
//
static TPM_RC RetrieveSessionData(
    COMMAND* command  // IN: main parsing structure for command
)
```

```
{
    int          i;
    TPM_RC       result;
    SESSION*     session;
    TPMA_SESSION sessionAttributes;
    TPM_HT       sessionType;
    INT32        sessionIndex;
    TPM_RC       errorIndex;
    //
    s_decryptSessionIndex = UNDEFINED_INDEX;
    s_encryptSessionIndex = UNDEFINED_INDEX;
    s_auditSessionIndex   = UNDEFINED_INDEX;

    for(sessionIndex = 0; command->authSize > 0; sessionIndex++)
    {
        errorIndex = TPM_RC_S + g_rcIndex[sessionIndex];

        // If maximum allowed number of sessions has been parsed, return a size
        // error with a session number that is larger than the number of allowed
        // sessions
        if(sessionIndex == MAX_SESSION_NUM)
            return TPM_RCS_SIZE + errorIndex;
        // make sure that the associated handle for each session starts out
        // unassigned
        s_associatedHandles[sessionIndex] = TPM_RH_UNASSIGNED;

        // First parameter: Session handle.
        result = TPMI_SH_AUTH_SESSION_Unmarshal(&s_sessionHandles[sessionIndex],
                                                &command->parameterBuffer,
                                                &command->authSize,
                                                TRUE);
        if(result != TPM_RC_SUCCESS)
            return result + TPM_RC_S + g_rcIndex[sessionIndex];
        // Second parameter: Nonce.
        result = TPM2B_NONCE_Unmarshal(&s_nonceCaller[sessionIndex],
                                       &command->parameterBuffer,
                                       &command->authSize);
        if(result != TPM_RC_SUCCESS)
            return result + TPM_RC_S + g_rcIndex[sessionIndex];
        // Third parameter: sessionAttributes.
        result = TPMA_SESSION_Unmarshal(&s_attributes[sessionIndex],
                                        &command->parameterBuffer,
                                        &command->authSize);
        if(result != TPM_RC_SUCCESS)
            return result + TPM_RC_S + g_rcIndex[sessionIndex];
        // Fourth parameter: authValue (PW or HMAC).
        result = TPM2B_AUTH_Unmarshal(&s_inputAuthValues[sessionIndex],
                                      &command->parameterBuffer,
                                      &command->authSize);
        if(result != TPM_RC_SUCCESS)
            return result + errorIndex;

        sessionAttributes = s_attributes[sessionIndex];
        if(s_sessionHandles[sessionIndex] == TPM_RS_PW)
        {
            // A PWAP session needs additional processing.
            //     Can't have any attributes set other than continueSession bit
            if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, encrypt)
               || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, decrypt)
               || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, audit)
               || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditExclusive)
               || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditReset))
                return TPM_RCS_ATTRIBUTES + errorIndex;
            //     The nonce size must be zero.
            if(s_nonceCaller[sessionIndex].t.size != 0)
                return TPM_RCS_NONCE + errorIndex;
```

```
        continue;
    }
    // For not password sessions...
    // Find out if the session is loaded.
    if(!SessionIsLoaded(s_sessionHandles[sessionIndex]))
        return TPM_RC_REFERENCE_S0 + sessionIndex;
    sessionType = HandleGetType(s_sessionHandles[sessionIndex]);
    session     = SessionGet(s_sessionHandles[sessionIndex]);

    // Check if the session is an HMAC/policy session.
    if((session->attributes.isPolicy == SET && sessionType == TPM_HT_HMAC_SESSION)
       || (session->attributes.isPolicy == CLEAR
           && sessionType == TPM_HT_POLICY_SESSION))
        return TPM_RCS_HANDLE + errorIndex;
    // Check that this handle has not previously been used.
    for(i = 0; i < sessionIndex; i++)
    {
        if(s_sessionHandles[i] == s_sessionHandles[sessionIndex])
            return TPM_RCS_HANDLE + errorIndex;
    }
    // If the session is used for parameter encryption or audit as well, set
    // the corresponding Indexes.

    // First process decrypt.
    if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, decrypt))
    {
        // Check if the commandCode allows command parameter encryption.
        if(DecryptSize(command->index) == 0)
            return TPM_RCS_ATTRIBUTES + errorIndex;
        // Encrypt attribute can only appear in one session
        if(s_decryptSessionIndex != UNDEFINED_INDEX)
            return TPM_RCS_ATTRIBUTES + errorIndex;
        // Can't decrypt if the session's symmetric algorithm is TPM_ALG_NULL
        if(session->symmetric.algorithm == TPM_ALG_NULL)
            return TPM_RCS_SYMMETRIC + errorIndex;
        // All checks passed, so set the index for the session used to decrypt
        // a command parameter.
        s_decryptSessionIndex = sessionIndex;
    }
    // Now process encrypt.
    if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, encrypt))
    {
        // Check if the commandCode allows response parameter encryption.
        if(EncryptSize(command->index) == 0)
            return TPM_RCS_ATTRIBUTES + errorIndex;
        // Encrypt attribute can only appear in one session.
        if(s_encryptSessionIndex != UNDEFINED_INDEX)
            return TPM_RCS_ATTRIBUTES + errorIndex;
        // Can't encrypt if the session's symmetric algorithm is TPM_ALG_NULL
        if(session->symmetric.algorithm == TPM_ALG_NULL)
            return TPM_RCS_SYMMETRIC + errorIndex;
        // All checks passed, so set the index for the session used to encrypt
        // a response parameter.
        s_encryptSessionIndex = sessionIndex;
    }
    // At last process audit.
    if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, audit))
    {
        // Audit attribute can only appear in one session.
        if(s_auditSessionIndex != UNDEFINED_INDEX)
            return TPM_RCS_ATTRIBUTES + errorIndex;
        // An audit session can not be policy session.
        if(HandleGetType(s_sessionHandles[sessionIndex]) == TPM_HT_POLICY_SESSION)
            return TPM_RCS_ATTRIBUTES + errorIndex;
        // If this is a reset of the audit session, or the first use
        // of the session as an audit session, it doesn't matter what
```

```
            // the exclusive state is. The session will become exclusive.
            if(!IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditReset)
               && session->attributes.isAudit == SET)
            {
                // Not first use or reset. If auditExlusive is SET, then this
                // session must be the current exclusive session.
                if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditExclusive)
                   && g_exclusiveAuditSession != s_sessionHandles[sessionIndex])
                    return TPM_RC_EXCLUSIVE;
            }
            s_auditSessionIndex = sessionIndex;
        }
        // Initialize associated handle as undefined. This will be changed when
        // the handles are processed.
        s_associatedHandles[sessionIndex] = TPM_RH_UNASSIGNED;
    }
    command->sessionNum = sessionIndex;
    return TPM_RC_SUCCESS;
}

//*** CheckLockedOut()
// This function checks to see if the TPM is in lockout. This function should only
// be called if the entity being checked is subject to DA protection. The TPM
// is in lockout if the NV is not available and a DA write is pending. Otherwise
// the TPM is locked out if checking for lockoutAuth ('lockoutAuthCheck' == TRUE)
// and use of lockoutAuth is disabled, or 'failedTries' >= 'maxTries'
//  Return Type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting
//      TPM_RC_NV_UNAVAILABLE   NV is not available at this time
//      TPM_RC_LOCKOUT          TPM is in lockout
static TPM_RC CheckLockedOut(
    BOOL lockoutAuthCheck  // IN: TRUE if checking is for lockoutAuth
)
{
    // If NV is unavailable, and current cycle state recorded in NV is not
    // SU_NONE_VALUE, refuse to check any authorization because we would
    // not be able to handle a DA failure.
    if(!NV_IS_AVAILABLE && NV_IS_ORDERLY)
        return g_NvStatus;
    // Check if DA info needs to be updated in NV.
    if(s_DAPendingOnNV)
    {
        // If NV is accessible,
        RETURN_IF_NV_IS_NOT_AVAILABLE;

        // ... write the pending DA data and proceed.
        NV_SYNC_PERSISTENT(lockOutAuthEnabled);
        NV_SYNC_PERSISTENT(failedTries);
        s_DAPendingOnNV = FALSE;
    }
    // Lockout is in effect if checking for lockoutAuth and use of lockoutAuth
    // is disabled...
    if(lockoutAuthCheck)
    {
        if(gp.lockOutAuthEnabled == FALSE)
            return TPM_RC_LOCKOUT;
    }
    else
    {
        // ... or if the number of failed tries has been maxed out.
        if(gp.failedTries >= gp.maxTries)
            return TPM_RC_LOCKOUT;
#if USE_DA_USED
        // If the daUsed flag is not SET, then no DA validation until the
        // daUsed state is written to NV
        if(!g_daUsed)
```

```
        {
            RETURN_IF_NV_IS_NOT_AVAILABLE;
            g_daUsed        = TRUE;
            gp.orderlyState = SU_DA_USED_VALUE;
            NV_SYNC_PERSISTENT(orderlyState);
            return TPM_RC_RETRY;
        }
#endif
    }
    return TPM_RC_SUCCESS;
}

//*** CheckAuthSession()
// This function checks that the authorization session properly authorizes the
// use of the associated handle.
//
//  Return Type: TPM_RC
//      TPM_RC_LOCKOUT              entity is protected by DA and TPM is in
//                                  lockout, or TPM is locked out on NV update
//                                  pending on DA parameters
//
//      TPM_RC_PP                   Physical Presence is required but not provided
//      TPM_RC_AUTH_FAIL            HMAC or PW authorization failed
//                                  with DA side-effects (can be a policy session)
//
//      TPM_RC_BAD_AUTH             HMAC or PW authorization failed without DA
//                                  side-effects (can be a policy session)
//
//      TPM_RC_POLICY_FAIL          if policy session fails
//      TPM_RC_POLICY_CC            command code of policy was wrong
//      TPM_RC_EXPIRED              the policy session has expired
//      TPM_RC_PCR
//      TPM_RC_AUTH_UNAVAILABLE     authValue or authPolicy unavailable
static TPM_RC CheckAuthSession(
    COMMAND* command,      // IN: primary parsing structure
    UINT32   sessionIndex  // IN: index of session to be processed
)
{
    TPM_RC     result            = TPM_RC_SUCCESS;
    SESSION*   session           = NULL;
    TPM_HANDLE sessionHandle     = s_sessionHandles[sessionIndex];
    TPM_HANDLE associatedHandle  = s_associatedHandles[sessionIndex];
    TPM_HT     sessionHandleType = HandleGetType(sessionHandle);
    BOOL       authUsed;
    //
    pAssert(sessionHandle != TPM_RH_UNASSIGNED);

    // Take care of physical presence
    if(associatedHandle == TPM_RH_PLATFORM)
    {
        // If the physical presence is required for this command, check for PP
        // assertion. If it isn't asserted, no point going any further.
        if(PhysicalPresenceIsRequired(command->index)
           && !_plat__PhysicalPresenceAsserted())
            return TPM_RC_PP;
    }
    if(sessionHandle != TPM_RS_PW)
    {
        session = SessionGet(sessionHandle);

        // Set includeAuth to indicate if DA checking will be required and if the
        // authValue will be included in any HMAC.
        if(sessionHandleType == TPM_HT_POLICY_SESSION)
        {
            // For a policy session, will check the DA status of the entity if either
            // isAuthValueNeeded or isPasswordNeeded is SET.
```

```
            session->attributes.includeAuth = session->attributes.isAuthValueNeeded
                                              || session->attributes.isPasswordNeeded;
        }
        else
        {
            // For an HMAC session, need to check unless the session
            // is bound.
            session->attributes.includeAuth =
                !IsSessionBindEntity(s_associatedHandles[sessionIndex], session);
        }
        authUsed = session->attributes.includeAuth;
    }
    else
        // Password session
        authUsed = TRUE;
    // If the authorization session is going to use an authValue, then make sure
    // that access to that authValue isn't locked out.
    if(authUsed)
    {
        // See if entity is subject to lockout.
        if(!IsDAExempted(associatedHandle))
        {
            // See if in lockout
            result = CheckLockedOut(associatedHandle == TPM_RH_LOCKOUT);
            if(result != TPM_RC_SUCCESS)
                return result;
        }
    }
    // Policy or HMAC+PW?
    if(sessionHandleType != TPM_HT_POLICY_SESSION)
    {
        // for non-policy session make sure that a policy session is not required
        if(IsPolicySessionRequired(command->index, sessionIndex))
            return TPM_RC_AUTH_TYPE;
        // The authValue must be available.
        // Note: The authValue is going to be "used" even if it is an EmptyAuth.
        // and the session is bound.
        if(!IsAuthValueAvailable(associatedHandle, command->index, sessionIndex))
            return TPM_RC_AUTH_UNAVAILABLE;
    }
    else
    {
        // ... see if the entity has a policy, ...
        // Note: IsAuthPolicyAvalable will return FALSE if the sensitive area of the
        // object is not loaded
        if(!IsAuthPolicyAvailable(associatedHandle, command->index, sessionIndex))
            return TPM_RC_AUTH_UNAVAILABLE;
        // ... and check the policy session.
        result = CheckPolicyAuthSession(command, sessionIndex);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    // Check authorization according to the type
    if((TPM_RS_PW == sessionHandle) || (session->attributes.isPasswordNeeded == SET))
        result = CheckPWAuthSession(sessionIndex);
    else
        result = CheckSessionHMAC(command, sessionIndex);
    // Do processing for PIN Indexes are only three possibilities for 'result' at
    // this point: TPM_RC_SUCCESS, TPM_RC_AUTH_FAIL, and TPM_RC_BAD_AUTH.
    // For all these cases, we would have to process a PIN index if the
    // authValue of the index was used for authorization.
    if((TPM_HT_NV_INDEX == HandleGetType(associatedHandle)) && authUsed)
    {
        NV_REF    locator;
        NV_INDEX* nvIndex = NvGetIndexInfo(associatedHandle, &locator);
        NV_PIN    pinData;
```

```
        TPMA_NV   nvAttributes;
        //
        pAssert(nvIndex != NULL);
        nvAttributes = nvIndex->publicArea.attributes;
        // If this is a PIN FAIL index and the value has been written
        // then we can update the counter (increment or clear)
        if(IsNvPinFailIndex(nvAttributes)
           && IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITTEN))
        {
            pinData.intVal = NvGetUINT64Data(nvIndex, locator);
            if(result != TPM_RC_SUCCESS)
                pinData.pin.pinCount++;
            else
                pinData.pin.pinCount = 0;
            NvWriteUINT64Data(nvIndex, pinData.intVal);
        }
        // If this is a PIN PASS Index, increment if we have used the
        // authorization value.
        // NOTE: If the counter has already hit the limit, then we
        // would not get here because the authorization value would not
        // be available and the TPM would have returned before it gets here
        else if(IsNvPinPassIndex(nvAttributes)
                && IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITTEN)
                && result == TPM_RC_SUCCESS)
        {
            // If the access is valid, then increment the use counter
            pinData.intVal = NvGetUINT64Data(nvIndex, locator);
            pinData.pin.pinCount++;
            NvWriteUINT64Data(nvIndex, pinData.intVal);
        }
    }
    return result;
}

#if CC_GetCommandAuditDigest
//*** CheckCommandAudit()
// This function is called before the command is processed if audit is enabled
// for the command. It will check to see if the audit can be performed and
// will ensure that the cpHash is available for the audit.
//  Return Type: TPM_RC
//      TPM_RC_NV_UNAVAILABLE       NV is not available for write
//      TPM_RC_NV_RATE              NV is rate limiting
static TPM_RC CheckCommandAudit(COMMAND* command)
{
    // If the audit digest is clear and command audit is required, NV must be
    // available so that TPM2_GetCommandAuditDigest() is able to increment
    // audit counter. If NV is not available, the function bails out to prevent
    // the TPM from attempting an operation that would fail anyway.
    if(gr.commandAuditDigest.t.size == 0
       || GetCommandCode(command->index) == TPM_CC_GetCommandAuditDigest)
    {
        RETURN_IF_NV_IS_NOT_AVAILABLE;
    }
    // Make sure that the cpHash is computed for the algorithm
    ComputeCpHash(command, gp.auditHashAlg);
    return TPM_RC_SUCCESS;
}
#endif

//*** ParseSessionBuffer()
// This function is the entry function for command session processing.
// It iterates sessions in session area and reports if the required authorization
// has been properly provided. It also processes audit session and passes the
// information of encryption sessions to parameter encryption module.
//
//  Return Type: TPM_RC
```

```
//        various           parsing failure or authorization failure
//
TPM_RC
ParseSessionBuffer(COMMAND* command  // IN: the structure that contains
)
{
    TPM_RC     result;
    UINT32     i;
    INT32      size = 0;
    TPM2B_AUTH extraKey;
    UINT32     sessionIndex;
    TPM_RC     errorIndex;
    SESSION*   session = NULL;
    //
    // Check if a command allows any session in its session area.
    if(!IsSessionAllowed(command->index))
        return TPM_RC_AUTH_CONTEXT;
    // Default-initialization.
    command->sessionNum = 0;

    result              = RetrieveSessionData(command);
    if(result != TPM_RC_SUCCESS)
        return result;
    // There is no command in the TPM spec that has more handles than
    // MAX_SESSION_NUM.
    pAssert(command->handleNum <= MAX_SESSION_NUM);

    // Associate the session with an authorization handle.
    for(i = 0; i < command->handleNum; i++)
    {
        if(CommandAuthRole(command->index, i) != AUTH_NONE)
        {
            // If the received session number is less than the number of handles
            // that requires authorization, an error should be returned.
            // Note: for all the TPM 2.0 commands, handles requiring
            // authorization come first in a command input and there are only ever
            // two values requiring authorization
            if(i > (command->sessionNum - 1))
                return TPM_RC_AUTH_MISSING;
            // Record the handle associated with the authorization session
            s_associatedHandles[i] = HierarchyNormalizeHandle(command->handles[i]);
        }
    }
    // Consistency checks are done first to avoid authorization failure when the
    // command will not be executed anyway.
    for(sessionIndex = 0; sessionIndex < command->sessionNum; sessionIndex++)
    {
        errorIndex = TPM_RC_S + g_rcIndex[sessionIndex];
        // PW session must be an authorization session
        if(s_sessionHandles[sessionIndex] == TPM_RS_PW)
        {
            if(s_associatedHandles[sessionIndex] == TPM_RH_UNASSIGNED)
                return TPM_RCS_HANDLE + errorIndex;
            // a password session can't be audit, encrypt or decrypt
            if(IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, audit)
               || IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, encrypt)
               || IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, decrypt))
                return TPM_RCS_ATTRIBUTES + errorIndex;
            session = NULL;
        }
        else
        {
            session = SessionGet(s_sessionHandles[sessionIndex]);

            // A trial session can not appear in session area, because it cannot
            // be used for authorization, audit or encrypt/decrypt.
```

```
            if(session->attributes.isTrialPolicy == SET)
                return TPM_RCS_ATTRIBUTES + errorIndex;

            // See if the session is bound to a DA protected entity
            // NOTE: Since a policy session is never bound, a policy is still
            // usable even if the object is DA protected and the TPM is in
            // lockout.
            if(session->attributes.isDaBound == SET)
            {
                result = CheckLockedOut(session->attributes.isLockoutBound == SET);
                if(result != TPM_RC_SUCCESS)
                    return result;
            }
            // If this session is for auditing, make sure the cpHash is computed.
            if(IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, audit))
                ComputeCpHash(command, session->authHashAlg);
        }

        // if the session has an associated handle, check the authorization
        if(s_associatedHandles[sessionIndex] != TPM_RH_UNASSIGNED)
        {
            result = CheckAuthSession(command, sessionIndex);
            if(result != TPM_RC_SUCCESS)
                return RcSafeAddToResult(result, errorIndex);
        }
        else
        {
            // a session that is not for authorization must either be encrypt,
            // decrypt, or audit
            if(!IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, audit)
               && !IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, encrypt)
               && !IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, decrypt))
                return TPM_RCS_ATTRIBUTES + errorIndex;

            // no authValue included in any of the HMAC computations
            pAssert(session != NULL);
            session->attributes.includeAuth = CLEAR;

            // check HMAC for encrypt/decrypt/audit only sessions
            result = CheckSessionHMAC(command, sessionIndex);
            if(result != TPM_RC_SUCCESS)
                return RcSafeAddToResult(result, errorIndex);
        }
    }
#if CC_GetCommandAuditDigest
    // Check if the command should be audited. Need to do this before any parameter
    // encryption so that the cpHash for the audit is correct
    if(CommandAuditIsRequired(command->index))
    {
        result = CheckCommandAudit(command);
        if(result != TPM_RC_SUCCESS)
            return result;  // No session number to reference
    }
#endif
    // Decrypt the first parameter if applicable. This should be the last operation
    // in session processing.
    // If the encrypt session is associated with a handle and the handle's
    // authValue is available, then authValue is concatenated with sessionKey to
    // generate encryption key, no matter if the handle is the session bound entity
    // or not.
    if(s_decryptSessionIndex != UNDEFINED_INDEX)
    {
        // If this is an authorization session, include the authValue in the
        // generation of the decryption key
        if(s_associatedHandles[s_decryptSessionIndex] != TPM_RH_UNASSIGNED)
        {
```

```
            EntityGetAuthValue(s_associatedHandles[s_decryptSessionIndex], &extraKey);
        }
        else
        {
            extraKey.b.size = 0;
        }
        size   = DecryptSize(command->index);
        result = CryptParameterDecryption(s_sessionHandles[s_decryptSessionIndex],
                                          &s_nonceCaller[s_decryptSessionIndex].b,
                                          command->parameterSize,
                                          (UINT16)size,
                                          &extraKey,
                                          command->parameterBuffer);
        if(result != TPM_RC_SUCCESS)
            return RcSafeAddToResult(result,
                                     TPM_RC_S + g_rcIndex[s_decryptSessionIndex]);
    }

    return TPM_RC_SUCCESS;
}

//*** CheckAuthNoSession()
// Function to process a command with no session associated.
// The function makes sure all the handles in the command require no authorization.
//
//  Return Type: TPM_RC
//      TPM_RC_AUTH_MISSING         failure - one or more handles require
//                                  authorization
TPM_RC
CheckAuthNoSession(COMMAND* command  // IN: command parsing structure
)
{
    UINT32 i;
#if CC_GetCommandAuditDigest
    TPM_RC result = TPM_RC_SUCCESS;
#endif
    //
    // Check if the command requires authorization
    for(i = 0; i < command->handleNum; i++)
    {
        if(CommandAuthRole(command->index, i) != AUTH_NONE)
            return TPM_RC_AUTH_MISSING;
    }
#if CC_GetCommandAuditDigest
    // Check if the command should be audited.
    if(CommandAuditIsRequired(command->index))
    {
        result = CheckCommandAudit(command);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
#endif
    // Initialize number of sessions to be 0
    command->sessionNum = 0;

    return TPM_RC_SUCCESS;
}

//** Response Session Processing
//*** Introduction
//
//  The following functions build the session area in a response and handle
//  the audit sessions (if present).
//

//*** ComputeRpHash()
```

```
// Function to compute rpHash (Response Parameter Hash). The rpHash is only
// computed if there is an HMAC authorization session and the return code is
// TPM_RC_SUCCESS.
static TPM2B_DIGEST* ComputeRpHash(
    COMMAND*   command,  // IN: command structure
    TPM_ALG_ID hashAlg   // IN: hash algorithm to compute rpHash
)
{
    TPM2B_DIGEST* rpHash = GetRpHashPointer(command, hashAlg);
    HASH_STATE    hashState;
    //
    if(rpHash->t.size == 0)
    {
        //   rpHash := hash(responseCode || commandCode || parameters)

        // Initiate hash creation.
        rpHash->t.size = CryptHashStart(&hashState, hashAlg);

        // Add hash constituents.
        CryptDigestUpdateInt(&hashState, sizeof(TPM_RC), TPM_RC_SUCCESS);
        CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), command->code);
        CryptDigestUpdate(
            &hashState, command->parameterSize, command->parameterBuffer);
        // Complete hash computation.
        CryptHashEnd2B(&hashState, &rpHash->b);
    }
    return rpHash;
}

//*** InitAuditSession()
// This function initializes the audit data in an audit session.
static void InitAuditSession(SESSION* session  // session to be initialized
)
{
    // Mark session as an audit session.
    session->attributes.isAudit = SET;

    // Audit session can not be bound.
    session->attributes.isBound = CLEAR;

    // Size of the audit log is the size of session hash algorithm digest.
    session->u2.auditDigest.t.size = CryptHashGetDigestSize(session->authHashAlg);

    // Set the original digest value to be 0.
    MemorySet(&session->u2.auditDigest.t.buffer, 0, session->u2.auditDigest.t.size);
    return;
}

//*** UpdateAuditDigest
// Function to update an audit digest
static void UpdateAuditDigest(
    COMMAND* command, TPMI_ALG_HASH hashAlg, TPM2B_DIGEST* digest)
{
    HASH_STATE    hashState;
    TPM2B_DIGEST* cpHash = GetCpHash(command, hashAlg);
    TPM2B_DIGEST* rpHash = ComputeRpHash(command, hashAlg);
    //
    pAssert(cpHash != NULL);

    // digestNew :=  hash (digestOld || cpHash || rpHash)
    // Start hash computation.
    digest->t.size = CryptHashStart(&hashState, hashAlg);
    // Add old digest.
    CryptDigestUpdate2B(&hashState, &digest->b);
    // Add cpHash
    CryptDigestUpdate2B(&hashState, &cpHash->b);
```

```
    // Add rpHash
    CryptDigestUpdate2B(&hashState, &rpHash->b);
    // Finalize the hash.
    CryptHashEnd2B(&hashState, &digest->b);
}

//*** Audit()
//This function updates the audit digest in an audit session.
static void Audit(COMMAND* command,      // IN: primary control structure
                  SESSION* auditSession  // IN: loaded audit session
)
{
    UpdateAuditDigest(
        command, auditSession->authHashAlg, &auditSession->u2.auditDigest);
    return;
}

#if CC_GetCommandAuditDigest
//*** CommandAudit()
// This function updates the command audit digest.
static void CommandAudit(COMMAND* command  // IN:
)
{
    // If the digest.size is one, it indicates the special case of changing
    // the audit hash algorithm. For this case, no audit is done on exit.
    // NOTE: When the hash algorithm is changed, g_updateNV is set in order to
    // force an update to the NV on exit so that the change in digest will
    // be recorded. So, it is safe to exit here without setting any flags
    // because the digest change will be written to NV when this code exits.
    if(gr.commandAuditDigest.t.size == 1)
    {
        gr.commandAuditDigest.t.size = 0;
        return;
    }
    // If the digest size is zero, need to start a new digest and increment
    // the audit counter.
    if(gr.commandAuditDigest.t.size == 0)
    {
        gr.commandAuditDigest.t.size = CryptHashGetDigestSize(gp.auditHashAlg);
        MemorySet(gr.commandAuditDigest.t.buffer, 0, gr.commandAuditDigest.t.size);

        // Bump the counter and save its value to NV.
        gp.auditCounter++;
        NV_SYNC_PERSISTENT(auditCounter);
    }
    UpdateAuditDigest(command, gp.auditHashAlg, &gr.commandAuditDigest);
    return;
}
#endif

//*** UpdateAuditSessionStatus()
// This function updates the internal audit related states of a session. It will:
//  a) initialize the session as audit session and set it to be exclusive if this
//     is the first time it is used for audit or audit reset was requested;
//  b) report exclusive audit session;
//  c) extend audit log; and
//  d) clear exclusive audit session if no audit session found in the command.
static void UpdateAuditSessionStatus(
    COMMAND* command  // IN: primary control structure
)
{
    UINT32     i;
    TPM_HANDLE auditSession = TPM_RH_UNASSIGNED;
    //
    // Iterate through sessions
    for(i = 0; i < command->sessionNum; i++)
```

```
    {
        SESSION* session;
        //
        // PW session do not have a loaded session and can not be an audit
        // session either.  Skip it.
        if(s_sessionHandles[i] == TPM_RS_PW)
            continue;
        session = SessionGet(s_sessionHandles[i]);

        // If a session is used for audit
        if(IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, audit))
        {
            // An audit session has been found
            auditSession = s_sessionHandles[i];

            // If the session has not been an audit session yet, or
            // the auditSetting bits indicate a reset, initialize it and set
            // it to be the exclusive session
            if(session->attributes.isAudit == CLEAR
               || IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, auditReset))
            {
                InitAuditSession(session);
                g_exclusiveAuditSession = auditSession;
            }
            else
            {
                // Check if the audit session is the current exclusive audit
                // session and, if not, clear previous exclusive audit session.
                if(g_exclusiveAuditSession != auditSession)
                    g_exclusiveAuditSession = TPM_RH_UNASSIGNED;
            }
            // Report audit session exclusivity.
            if(g_exclusiveAuditSession == auditSession)
            {
                SET_ATTRIBUTE(s_attributes[i], TPMA_SESSION, auditExclusive);
            }
            else
            {
                CLEAR_ATTRIBUTE(s_attributes[i], TPMA_SESSION, auditExclusive);
            }
            // Extend audit log.
            Audit(command, session);
        }
    }
    // If no audit session is found in the command, and the command allows
    // a session then, clear the current exclusive
    // audit session.
    if(auditSession == TPM_RH_UNASSIGNED && IsSessionAllowed(command->index))
    {
        g_exclusiveAuditSession = TPM_RH_UNASSIGNED;
    }
    return;
}

//*** ComputeResponseHMAC()
// Function to compute HMAC for authorization session in a response.
/*(See part 1 specification)
// Function: Compute HMAC for response sessions
//      The sessionAuth value
//          authHMAC := HMACsHASH((sessionAuth | authValue),
//                      (pHash | nonceTPM | nonceCaller | sessionAttributes))
//  Where:
//      HMACsHASH()     The HMAC algorithm using the hash algorithm specified when
//                      the session was started.
//
//      sessionAuth     A TPMB_MEDIUM computed in a protocol-dependent way, using
```

```
//                      KDFa. In an HMAC or KDF, only sessionAuth.buffer is used.
//
//      authValue       A TPM2B_AUTH that is found in the sensitive area of an
//                      object. In an HMAC or KDF, only authValue.buffer is used
//                      and all trailing zeros are removed.
//
//      pHash           Response parameters (rpHash) using the session hash. When
//                      using a pHash in an HMAC computation, both the algorithm ID
//                      and the digest are included.
//
//      nonceTPM        A TPM2B_NONCE that is generated by the entity using the
//                      session. In an HMAC or KDF, only nonceTPM.buffer is used.
//
//      nonceCaller     a TPM2B_NONCE that was received the previous time the
//                      session was used. In an HMAC or KDF, only
//                      nonceCaller.buffer is used.
//
//      sessionAttributes   A TPMA_SESSION that indicates the attributes associated
//                          with a particular use of the session.
*/
static void ComputeResponseHMAC(
    COMMAND*      command,       // IN: command structure
    UINT32        sessionIndex,  // IN: session index to be processed
    SESSION*      session,       // IN: loaded session
    TPM2B_DIGEST* hmac           // OUT: authHMAC
)
{
    TPM2B_TYPE(KEY, (sizeof(AUTH_VALUE) * 2));
    TPM2B_KEY     key;  // HMAC key
    BYTE          marshalBuffer[sizeof(TPMA_SESSION)];
    BYTE*         buffer;
    UINT32        marshalSize;
    HMAC_STATE    hmacState;
    TPM2B_DIGEST* rpHash = ComputeRpHash(command, session->authHashAlg);
    //
    // Generate HMAC key
    MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));

    // Add the object authValue if required
    if(session->attributes.includeAuth == SET)
    {
        // Note: includeAuth may be SET for a policy that is used in
        // UndefineSpaceSpecial(). At this point, the Index has been deleted
        // so the includeAuth will have no meaning. However, the
        // s_associatedHandles[] value for the session is now set to TPM_RH_NULL so
        // this will return the authValue associated with TPM_RH_NULL and that is
        // and empty buffer.
        TPM2B_AUTH authValue;
        //
        // Get the authValue with trailing zeros removed
        EntityGetAuthValue(s_associatedHandles[sessionIndex], &authValue);

        // Add it to the key
        MemoryConcat2B(&key.b, &authValue.b, sizeof(key.t.buffer));
    }

    // if the HMAC key size is 0, the response HMAC is computed according to the
    // input HMAC
    if(key.t.size == 0 && s_inputAuthValues[sessionIndex].t.size == 0)
    {
        hmac->t.size = 0;
        return;
    }
    // Start HMAC computation.
    hmac->t.size = CryptHmacStart2B(&hmacState, session->authHashAlg, &key.b);
```

```
    // Add hash components.
    CryptDigestUpdate2B(&hmacState.hashState, &rpHash->b);
    CryptDigestUpdate2B(&hmacState.hashState, &session->nonceTPM.b);
    CryptDigestUpdate2B(&hmacState.hashState, &s_nonceCaller[sessionIndex].b);

    // Add session attributes.
    buffer      = marshalBuffer;
    marshalSize = TPMA_SESSION_Marshal(&s_attributes[sessionIndex], &buffer, NULL);
    CryptDigestUpdate(&hmacState.hashState, marshalSize, marshalBuffer);

    // Finalize HMAC.
    CryptHmacEnd2B(&hmacState, &hmac->b);

    return;
}

//*** UpdateInternalSession()
// This function updates internal sessions by:
// a) restarting session time; and
// b) clearing a policy session since nonce is rolling.
static void UpdateInternalSession(SESSION* session,  // IN: the session structure
                                  UINT32   i         // IN: session number
)
{
    // If nonce is rolling in a policy session, the policy related data
    // will be re-initialized.
    if(HandleGetType(s_sessionHandles[i]) == TPM_HT_POLICY_SESSION
       && IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, continueSession))
    {
        // When the nonce rolls it starts a new timing interval for the
        // policy session.
        SessionResetPolicyData(session);
        SessionSetStartTime(session);
    }
    return;
}

//*** BuildSingleResponseAuth()
//   Function to compute response HMAC value for a policy or HMAC session.
static TPM2B_NONCE* BuildSingleResponseAuth(
    COMMAND*    command,       // IN: command structure
    UINT32      sessionIndex,  // IN: session index to be processed
    TPM2B_AUTH* auth           // OUT: authHMAC
)
{
    // Fill in policy/HMAC based session response.
    SESSION* session = SessionGet(s_sessionHandles[sessionIndex]);
    //
    // If the session is a policy session with isPasswordNeeded SET, the
    // authorization field is empty.
    if(HandleGetType(s_sessionHandles[sessionIndex]) == TPM_HT_POLICY_SESSION
       && session->attributes.isPasswordNeeded == SET)
        auth->t.size = 0;
    else
        // Compute response HMAC.
        ComputeResponseHMAC(command, sessionIndex, session, auth);

    UpdateInternalSession(session, sessionIndex);
    return &session->nonceTPM;
}

//*** UpdateAllNonceTPM()
// Updates TPM nonce for all sessions in command.
static void UpdateAllNonceTPM(COMMAND* command  // IN: controlling structure
)
{
```

```
    UINT32   i;
    SESSION* session;
    //
    for(i = 0; i < command->sessionNum; i++)
    {
        // If not a PW session, compute the new nonceTPM.
        if(s_sessionHandles[i] != TPM_RS_PW)
        {
            session = SessionGet(s_sessionHandles[i]);
            // Update nonceTPM in both internal session and response.
            CryptRandomGenerate(session->nonceTPM.t.size, session->nonceTPM.t.buffer);
        }
    }
    return;
}

//*** BuildResponseSession()
// Function to build Session buffer in a response. The authorization data is added
// to the end of command->responseBuffer. The size of the authorization area is
// accumulated in command->authSize.
// When this is called, command->responseBuffer is pointing at the next location
// in the response buffer to be filled. This is where the authorization sessions
// will go, if any. command->parameterSize is the number of bytes that have been
// marshaled as parameters in the output buffer.
TPM_RC
BuildResponseSession(COMMAND* command  // IN: structure that has relevant command
                                       //     information
)
{
    TPM_RC result = TPM_RC_SUCCESS;

    pAssert(command->authSize == 0);

    // Reset the parameter buffer to point to the start of the parameters so that
    // there is a starting point for any rpHash that might be generated and so there
    // is a place where parameter encryption would start
    command->parameterBuffer = command->responseBuffer - command->parameterSize;

    // Session nonces should be updated before parameter encryption
    if(command->tag == TPM_ST_SESSIONS)
    {
        UpdateAllNonceTPM(command);

        // Encrypt first parameter if applicable. Parameter encryption should
        // happen after nonce update and before any rpHash is computed.
        // If the encrypt session is associated with a handle, the authValue of
        // this handle will be concatenated with sessionKey to generate
        // encryption key, no matter if the handle is the session bound entity
        // or not. The authValue is added to sessionKey only when the authValue
        // is available.
        if(s_encryptSessionIndex != UNDEFINED_INDEX)
        {
            UINT32     size;
            TPM2B_AUTH extraKey;
            //
            extraKey.b.size = 0;
            // If this is an authorization session, include the authValue in the
            // generation of the encryption key
            if(s_associatedHandles[s_encryptSessionIndex] != TPM_RH_UNASSIGNED)
            {
                EntityGetAuthValue(s_associatedHandles[s_encryptSessionIndex],
                                   &extraKey);
            }
            size = EncryptSize(command->index);
            // This function operates on internally-generated data that is
            // expected to be well-formed for parameter encryption.
```

```
            // In the event that there is a bug elsewhere in the code and the
            // input data is not well-formed, CryptParameterEncryption will
            // put the TPM into failure mode instead of allowing the out-of-
            // band write.
            CryptParameterEncryption(s_sessionHandles[s_encryptSessionIndex],
                                     &s_nonceCaller[s_encryptSessionIndex].b,
                                     command->parameterSize,
                                     (UINT16)size,
                                     &extraKey,
                                     command->parameterBuffer);
            if(g_inFailureMode)
            {
                result = TPM_RC_FAILURE;
                goto Cleanup;
            }
        }
    }
    // Audit sessions should be processed regardless of the tag because
    // a command with no session may cause a change of the exclusivity state.
    UpdateAuditSessionStatus(command);
#if CC_GetCommandAuditDigest
    // Command Audit
    if(CommandAuditIsRequired(command->index))
        CommandAudit(command);
#endif
    // Process command with sessions.
    if(command->tag == TPM_ST_SESSIONS)
    {
        UINT32 i;
        //
        pAssert(command->sessionNum > 0);

        // Iterate over each session in the command session area, and create
        // corresponding sessions for response.
        for(i = 0; i < command->sessionNum; i++)
        {
            TPM2B_NONCE* nonceTPM;
            TPM2B_DIGEST responseAuth;
            // Make sure that continueSession is SET on any Password session.
            // This makes it marginally easier for the management software
            // to keep track of the closed sessions.
            if(s_sessionHandles[i] == TPM_RS_PW)
            {
                SET_ATTRIBUTE(s_attributes[i], TPMA_SESSION, continueSession);
                responseAuth.t.size = 0;
                nonceTPM            = (TPM2B_NONCE*)&responseAuth;
            }
            else
            {
                // Compute the response HMAC and get a pointer to the nonce used.
                // This function will also update the values if needed. Note, the
                nonceTPM = BuildSingleResponseAuth(command, i, &responseAuth);
            }
            command->authSize +=
                TPM2B_NONCE_Marshal(nonceTPM, &command->responseBuffer, NULL);
            command->authSize += TPMA_SESSION_Marshal(
                &s_attributes[i], &command->responseBuffer, NULL);
            command->authSize +=
                TPM2B_DIGEST_Marshal(&responseAuth, &command->responseBuffer, NULL);
            if(!IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, continueSession))
                SessionFlush(s_sessionHandles[i]);
        }
    }

Cleanup:
    return result;
```

```
}

//*** SessionRemoveAssociationToHandle()
// This function deals with the case where an entity associated with an authorization
// is deleted during command processing. The primary use of this is to support
// UndefineSpaceSpecial().
void SessionRemoveAssociationToHandle(TPM_HANDLE handle)
{
    UINT32 i;
    //
    for(i = 0; i < MAX_SESSION_NUM; i++)
    {
        if(s_associatedHandles[i] == HierarchyNormalizeHandle(handle))
        {
            s_associatedHandles[i] = TPM_RH_NULL;
        }
    }
}
```

7.170 /tpm/src/subsystem/CommandAudit.c

```
//** Introduction
// This file contains the functions that support command audit.

//** Includes
#include "Tpm.h"

//** Functions

//*** CommandAuditPreInstall_Init()
// This function initializes the command audit list. This function simulates
// the behavior of manufacturing. A function is used instead of a structure
// definition because this is easier than figuring out the initialization value
// for a bit array.
//
// This function would not be implemented outside of a manufacturing or
// simulation environment.
void CommandAuditPreInstall_Init(void)
{
    // Clear all the audit commands
    MemorySet(gp.auditCommands, 0x00, sizeof(gp.auditCommands));

    // TPM_CC_SetCommandCodeAuditStatus always being audited
    CommandAuditSet(TPM_CC_SetCommandCodeAuditStatus);

    // Set initial command audit hash algorithm to be context integrity hash
    // algorithm
    gp.auditHashAlg = CONTEXT_INTEGRITY_HASH_ALG;

    // Set up audit counter to be 0
    gp.auditCounter = 0;

    // Write command audit persistent data to NV
    NV_SYNC_PERSISTENT(auditCommands);
    NV_SYNC_PERSISTENT(auditHashAlg);
    NV_SYNC_PERSISTENT(auditCounter);

    return;
}

//*** CommandAuditStartup()
// This function clears the command audit digest on a TPM Reset.
BOOL CommandAuditStartup(STARTUP_TYPE type  // IN: start up type
)
{
```

```
    if((type != SU_RESTART) && (type != SU_RESUME))
    {
        // Reset the digest size to initialize the digest
        gr.commandAuditDigest.t.size = 0;
    }
    return TRUE;
}

//*** CommandAuditSet()
// This function will SET the audit flag for a command. This function
// will not SET the audit flag for a command that is not implemented. This
// ensures that the audit status is not SET when TPM2_GetCapability() is
// used to read the list of audited commands.
//
// This function is only used by TPM2_SetCommandCodeAuditStatus().
//
// The actions in TPM2_SetCommandCodeAuditStatus() are expected to cause the
// changes to be saved to NV after it is setting and clearing bits.
//  Return Type: BOOL
//      TRUE(1)         command code audit status was changed
//      FALSE(0)        command code audit status was not changed
BOOL CommandAuditSet(TPM_CC commandCode  // IN: command code
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);

    // Only SET a bit if the corresponding command is implemented
    if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
    {
        // Can't audit shutdown
        if(commandCode != TPM_CC_Shutdown)
        {
            if(!TEST_BIT(commandIndex, gp.auditCommands))
            {
                // Set bit
                SET_BIT(commandIndex, gp.auditCommands);
                return TRUE;
            }
        }
    }
    // No change
    return FALSE;
}

//*** CommandAuditClear()
// This function will CLEAR the audit flag for a command. It will not CLEAR the
// audit flag for TPM_CC_SetCommandCodeAuditStatus().
//
// This function is only used by TPM2_SetCommandCodeAuditStatus().
//
// The actions in TPM2_SetCommandCodeAuditStatus() are expected to cause the
// changes to be saved to NV after it is setting and clearing bits.
//  Return Type: BOOL
//      TRUE(1)         command code audit status was changed
//      FALSE(0)        command code audit status was not changed
BOOL CommandAuditClear(TPM_CC commandCode  // IN: command code
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);

    // Do nothing if the command is not implemented
    if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
    {
        // The bit associated with TPM_CC_SetCommandCodeAuditStatus() cannot be
        // cleared
        if(commandCode != TPM_CC_SetCommandCodeAuditStatus)
```

```
        {
            if(TEST_BIT(commandIndex, gp.auditCommands))
            {
                // Clear bit
                CLEAR_BIT(commandIndex, gp.auditCommands);
                return TRUE;
            }
        }
    }
    // No change
    return FALSE;
}

//*** CommandAuditIsRequired()
// This function indicates if the audit flag is SET for a command.
//  Return Type: BOOL
//      TRUE(1)         command is audited
//      FALSE(0)        command is not audited
BOOL CommandAuditIsRequired(COMMAND_INDEX commandIndex  // IN: command index
)
{
    // Check the bit map.  If the bit is SET, command audit is required
    return (TEST_BIT(commandIndex, gp.auditCommands));
}

//*** CommandAuditCapGetCCList()
// This function returns a list of commands that have their audit bit SET.
//
// The list starts at the input commandCode.
//  Return Type: TPMI_YES_NO
//      YES         if there are more command code available
//      NO          all the available command code has been returned
TPMI_YES_NO
CommandAuditCapGetCCList(TPM_CC   commandCode,  // IN: start command code
                         UINT32   count,        // IN: count of returned TPM_CC
                         TPML_CC* commandList   // OUT: list of TPM_CC
)
{
    TPMI_YES_NO   more = NO;
    COMMAND_INDEX commandIndex;

    // Initialize output handle list
    commandList->count = 0;

    // The maximum count of command we may return is MAX_CAP_CC
    if(count > MAX_CAP_CC)
        count = MAX_CAP_CC;

    // Find the implemented command that has a command code that is the same or
    // higher than the input
    // Collect audit commands
    for(commandIndex = GetClosestCommandIndex(commandCode);
        commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
        commandIndex = GetNextCommandIndex(commandIndex))
    {
        if(CommandAuditIsRequired(commandIndex))
        {
            if(commandList->count < count)
            {
                // If we have not filled up the return list, add this command
                // code to its
                TPM_CC cc =
                    GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex);
                if(IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))
                    cc += (1 << 29);
                commandList->commandCodes[commandList->count] = cc;
```

```
                commandList->count++;
            }
            else
            {
                // If the return list is full but we still have command
                // available, report this and stop iterating
                more = YES;
                break;
            }
        }
    }

    return more;
}

//*** CommandAuditCapGetOneCC()
// This function returns true if a command has its audit bit set.
BOOL CommandAuditCapGetOneCC(TPM_CC commandCode)  // IN: command code
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);
    if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
    {
        return CommandAuditIsRequired(commandIndex);
    }
    return FALSE;
}

//*** CommandAuditGetDigest
// This command is used to create a digest of the commands being audited. The
// commands are processed in ascending numeric order with a list of TPM_CC being
// added to a hash. This operates as if all the audited command codes were
// concatenated and then hashed.
void CommandAuditGetDigest(TPM2B_DIGEST* digest  // OUT: command digest
)
{
    TPM_CC        commandCode;
    COMMAND_INDEX commandIndex;
    HASH_STATE    hashState;

    // Start hash
    digest->t.size = CryptHashStart(&hashState, gp.auditHashAlg);

    // Add command code
    for(commandIndex = 0; commandIndex < COMMAND_COUNT; commandIndex++)
    {
        if(CommandAuditIsRequired(commandIndex))
        {
            commandCode = GetCommandCode(commandIndex);
            CryptDigestUpdateInt(&hashState, sizeof(commandCode), commandCode);
        }
    }

    // Complete hash
    CryptHashEnd2B(&hashState, &digest->b);

    return;
}
```

7.171 /tpm/src/subsystem/DA.c

```
//** Introduction
// This file contains the functions and data definitions relating to the
// dictionary attack logic.

//** Includes and Data Definitions
```

```
#define DA_C
#include "Tpm.h"

//** Functions

//*** DAPreInstall_Init()
// This function initializes the DA parameters to their manufacturer-default
// values. The default values are determined by a platform-specific specification.
//
// This function should not be called outside of a manufacturing or simulation
// environment.
//
// The DA parameters will be restored to these initial values by TPM2_Clear().
void DAPreInstall_Init(void)
{
    gp.failedTries        = 0;
    gp.maxTries           = 3;
    gp.recoveryTime       = 1000;  // in seconds (~16.67 minutes)
    gp.lockoutRecovery    = 1000;  // in seconds
    gp.lockOutAuthEnabled = TRUE;  // Use of lockoutAuth is enabled

    // Record persistent DA parameter changes to NV
    NV_SYNC_PERSISTENT(failedTries);
    NV_SYNC_PERSISTENT(maxTries);
    NV_SYNC_PERSISTENT(recoveryTime);
    NV_SYNC_PERSISTENT(lockoutRecovery);
    NV_SYNC_PERSISTENT(lockOutAuthEnabled);

    return;
}

//*** DAStartup()
// This function is called  by TPM2_Startup() to initialize the DA parameters.
// In the case of Startup(CLEAR), use of lockoutAuth will be enabled if the
// lockout recovery time is 0. Otherwise, lockoutAuth will not be enabled until
// the TPM has been continuously powered for the lockoutRecovery time.
//
// This function requires that NV be available and not rate limiting.
BOOL DAStartup(STARTUP_TYPE type  // IN: startup type
)
{
    NOT_REFERENCED(type);
#if !ACCUMULATE_SELF_HEAL_TIMER
    _plat__TimerWasReset();
    s_selfHealTimer = 0;
    s_lockoutTimer  = 0;
#else
    if(_plat__TimerWasReset())
    {
        if(!NV_IS_ORDERLY)
        {
            // If shutdown was not orderly, then don't really know if go.time has
            // any useful value so reset the timer to 0. This is what the tick
            // was reset to
            s_selfHealTimer = 0;
            s_lockoutTimer  = 0;
        }
        else
        {
            // If we know how much time was accumulated at the last orderly shutdown
            // subtract that from the saved timer values so that they effectively
            // have the accumulated values
            s_selfHealTimer -= go.time;
            s_lockoutTimer -= go.time;
        }
    }
```

```
#endif

    // For any Startup(), if lockoutRecovery is 0, enable use of lockoutAuth.
    if(gp.lockoutRecovery == 0)
    {
        gp.lockOutAuthEnabled = TRUE;
        // Record the changes to NV
        NV_SYNC_PERSISTENT(lockOutAuthEnabled);
    }

    // If DA has not been disabled and the previous shutdown is not orderly
    // failedTries is not already at its maximum then increment 'failedTries'
    if(gp.recoveryTime != 0 && gp.failedTries < gp.maxTries
       && !IS_ORDERLY(g_prevOrderlyState))
    {
#if USE_DA_USED
        gp.failedTries += g_daUsed;
        g_daUsed = FALSE;
#else
        gp.failedTries++;
#endif
        // Record the change to NV
        NV_SYNC_PERSISTENT(failedTries);
    }
    // Before Startup, the TPM will not do clock updates. At startup, need to
    // do a time update which will do the DA update.
    TimeUpdate();

    return TRUE;
}

//*** DARegisterFailure()
// This function is called when an authorization failure occurs on an entity
// that is subject to dictionary-attack protection. When a DA failure is
// triggered, register the failure by resetting the relevant self-healing
// timer to the current time.
void DARegisterFailure(TPM_HANDLE handle  // IN: handle for failure
)
{
    // Reset the timer associated with lockout if the handle is the lockoutAuth.
    if(handle == TPM_RH_LOCKOUT)
        s_lockoutTimer = g_time;
    else
        s_selfHealTimer = g_time;
    return;
}

//*** DASelfHeal()
// This function is called to check if sufficient time has passed to allow
// decrement of failedTries or to re-enable use of lockoutAuth.
//
// This function should be called when the time interval is updated.
void DASelfHeal(void)
{
    // Regular authorization self healing logic
    // If no failed authorization tries, do nothing.  Otherwise, try to
    // decrease failedTries
    if(gp.failedTries != 0)
    {
        // if recovery time is 0, DA logic has been disabled.  Clear failed tries
        // immediately
        if(gp.recoveryTime == 0)
        {
            gp.failedTries = 0;
            // Update NV record
            NV_SYNC_PERSISTENT(failedTries);
```

```
        }
        else
        {
            UINT64 decreaseCount;
#if 0
            // Errata eliminates this code
            // In the unlikely event that failedTries should become larger than
            // maxTries
            if(gp.failedTries > gp.maxTries)
                gp.failedTries = gp.maxTries;
#endif
            // How much can failedTries be decreased

            // Cast s_selfHealTimer to an int in case it became negative at
            // startup
            decreaseCount =
                ((g_time - (INT64)s_selfHealTimer) / 1000) / gp.recoveryTime;

            if(gp.failedTries <= (UINT32)decreaseCount)
                // should not set failedTries below zero
                gp.failedTries = 0;
            else
                gp.failedTries -= (UINT32)decreaseCount;

            // the cast prevents overflow of the product
            s_selfHealTimer += (decreaseCount * (UINT64)gp.recoveryTime) * 1000;
            if(decreaseCount != 0)
                // If there was a change to the failedTries, record the changes
                // to NV
                NV_SYNC_PERSISTENT(failedTries);
        }
    }

    // LockoutAuth self healing logic
    // If lockoutAuth is enabled, do nothing.  Otherwise, try to see if we
    // may enable it
    if(!gp.lockOutAuthEnabled)
    {
        // if lockout authorization recovery time is 0, a reboot is required to
        // re-enable use of lockout authorization.  Self-healing would not
        // apply in this case.
        if(gp.lockoutRecovery != 0)
        {
            if(((g_time - (INT64)s_lockoutTimer) / 1000) >= gp.lockoutRecovery)
            {
                gp.lockOutAuthEnabled = TRUE;
                // Record the changes to NV
                NV_SYNC_PERSISTENT(lockOutAuthEnabled);
            }
        }
    }
    return;
}
```

7.172 /tpm/src/subsystem/Hierarchy.c

```
//** Introduction
// This file contains the functions used for managing and accessing the
// hierarchy-related values.

//** Includes

#include "Tpm.h"

//**HIERARCHY_MODIFIER_TYPE
```

```
// This enumerates the possible hierarchy modifiers.
typedef enum
{
    HM_NONE = 0,
    HM_FW_LIMITED,  // Hierarchy is firmware-limited.
    HM_SVN_LIMITED  // Hierarchy is SVN-limited.
} HIERARCHY_MODIFIER_TYPE;

//*** HIERARCHY_MODIFIER Structure
// A HIERARCHY_MODIFIER structure holds metadata about an OBJECT's
// hierarchy modifier.
typedef struct HIERARCHY_MODIFIER
{
    HIERARCHY_MODIFIER_TYPE type;  // The type of modification.
    uint16_t min_svn;  // The minimum SVN to which the hierarchy is limited.
                       // Only valid if 'type' is HM_SVN_LIMITED.
} HIERARCHY_MODIFIER;

//** Functions

//*** HierarchyPreInstall()
// This function performs the initialization functions for the hierarchy
// when the TPM is simulated. This function should not be called if the
// TPM is not in a manufacturing mode at the manufacturer, or in a simulated
// environment.
void HierarchyPreInstall_Init(void)
{
    // Allow lockout clear command
    gp.disableClear = FALSE;

    // Initialize Primary Seeds
    gp.EPSeed.t.size = sizeof(gp.EPSeed.t.buffer);
    gp.SPSeed.t.size = sizeof(gp.SPSeed.t.buffer);
    gp.PPSeed.t.size = sizeof(gp.PPSeed.t.buffer);
#if(defined USE_PLATFORM_EPS) && (USE_PLATFORM_EPS != NO)
    _plat__GetEPS(gp.EPSeed.t.size, gp.EPSeed.t.buffer);
#else
    CryptRandomGenerate(gp.EPSeed.t.size, gp.EPSeed.t.buffer);
#endif
    CryptRandomGenerate(gp.SPSeed.t.size, gp.SPSeed.t.buffer);
    CryptRandomGenerate(gp.PPSeed.t.size, gp.PPSeed.t.buffer);

    // Initialize owner, endorsement and lockout authorization
    gp.ownerAuth.t.size       = 0;
    gp.endorsementAuth.t.size = 0;
    gp.lockoutAuth.t.size     = 0;

    // Initialize owner, endorsement, and lockout policy
    gp.ownerAlg                = TPM_ALG_NULL;
    gp.ownerPolicy.t.size      = 0;
    gp.endorsementAlg          = TPM_ALG_NULL;
    gp.endorsementPolicy.t.size = 0;
    gp.lockoutAlg              = TPM_ALG_NULL;
    gp.lockoutPolicy.t.size    = 0;

    // Initialize ehProof, shProof and phProof
    gp.phProof.t.size = sizeof(gp.phProof.t.buffer);
    gp.shProof.t.size = sizeof(gp.shProof.t.buffer);
    gp.ehProof.t.size = sizeof(gp.ehProof.t.buffer);
    CryptRandomGenerate(gp.phProof.t.size, gp.phProof.t.buffer);
    CryptRandomGenerate(gp.shProof.t.size, gp.shProof.t.buffer);
    CryptRandomGenerate(gp.ehProof.t.size, gp.ehProof.t.buffer);

    // Write hierarchy data to NV
    NV_SYNC_PERSISTENT(disableClear);
    NV_SYNC_PERSISTENT(EPSeed);
```

```
    NV_SYNC_PERSISTENT(SPSeed);
    NV_SYNC_PERSISTENT(PPSeed);
    NV_SYNC_PERSISTENT(ownerAuth);
    NV_SYNC_PERSISTENT(endorsementAuth);
    NV_SYNC_PERSISTENT(lockoutAuth);
    NV_SYNC_PERSISTENT(ownerAlg);
    NV_SYNC_PERSISTENT(ownerPolicy);
    NV_SYNC_PERSISTENT(endorsementAlg);
    NV_SYNC_PERSISTENT(endorsementPolicy);
    NV_SYNC_PERSISTENT(lockoutAlg);
    NV_SYNC_PERSISTENT(lockoutPolicy);
    NV_SYNC_PERSISTENT(phProof);
    NV_SYNC_PERSISTENT(shProof);
    NV_SYNC_PERSISTENT(ehProof);

    return;
}

//*** HierarchyStartup()
// This function is called at TPM2_Startup() to initialize the hierarchy
// related values.
BOOL HierarchyStartup(STARTUP_TYPE type  // IN: start up type
)
{
    // phEnable is SET on any startup
    g_phEnable = TRUE;

    // Reset platformAuth, platformPolicy; enable SH and EH at TPM_RESET and
    // TPM_RESTART
    if(type != SU_RESUME)
    {
        gc.platformAuth.t.size   = 0;
        gc.platformPolicy.t.size = 0;
        gc.platformAlg           = TPM_ALG_NULL;

        // enable the storage and endorsement hierarchies and the platformNV
        gc.shEnable = gc.ehEnable = gc.phEnableNV = TRUE;
    }

    // nullProof and nullSeed are updated at every TPM_RESET
    if((type != SU_RESTART) && (type != SU_RESUME))
    {
        gr.nullProof.t.size = sizeof(gr.nullProof.t.buffer);
        CryptRandomGenerate(gr.nullProof.t.size, gr.nullProof.t.buffer);
        gr.nullSeed.t.size = sizeof(gr.nullSeed.t.buffer);
        CryptRandomGenerate(gr.nullSeed.t.size, gr.nullSeed.t.buffer);
    }

    return TRUE;
}

//*** DecomposeHandle()
// This function extracts the base hierarchy and modifier from a given handle.
// Returns the base hierarchy.
static TPMI_RH_HIERARCHY DecomposeHandle(TPMI_RH_HIERARCHY   handle,   // IN
                                         HIERARCHY_MODIFIER* modifier  // OUT
)
{
    TPMI_RH_HIERARCHY base_hierarchy = handle;

    modifier->type                   = HM_NONE;

    // See if the handle is firmware-bound.
    switch(handle)
    {
        case TPM_RH_FW_OWNER:
```

```
        {
            modifier->type = HM_FW_LIMITED;
            base_hierarchy = TPM_RH_OWNER;
            break;
        }
        case TPM_RH_FW_ENDORSEMENT:
        {
            modifier->type = HM_FW_LIMITED;
            base_hierarchy = TPM_RH_ENDORSEMENT;
            break;
        }
        case TPM_RH_FW_PLATFORM:
        {
            modifier->type = HM_FW_LIMITED;
            base_hierarchy = TPM_RH_PLATFORM;
            break;
        }
        case TPM_RH_FW_NULL:
        {
            modifier->type = HM_FW_LIMITED;
            base_hierarchy = TPM_RH_NULL;
            break;
        }
    }

    if(modifier->type == HM_FW_LIMITED)
    {
        return base_hierarchy;
    }

    // See if the handle is SVN-bound.
    switch(handle & 0xFFFF0000)
    {
        case TPM_RH_SVN_OWNER_BASE:
            modifier->type = HM_SVN_LIMITED;
            base_hierarchy = TPM_RH_OWNER;
            break;
        case TPM_RH_SVN_ENDORSEMENT_BASE:
            modifier->type = HM_SVN_LIMITED;
            base_hierarchy = TPM_RH_ENDORSEMENT;
            break;
        case TPM_RH_SVN_PLATFORM_BASE:
            modifier->type = HM_SVN_LIMITED;
            base_hierarchy = TPM_RH_PLATFORM;
            break;
        case TPM_RH_SVN_NULL_BASE:
            modifier->type = HM_SVN_LIMITED;
            base_hierarchy = TPM_RH_NULL;
            break;
    }

    if(modifier->type == HM_SVN_LIMITED)
    {
        modifier->min_svn = handle & 0x0000FFFF;
        return base_hierarchy;
    }

    // Handle is neither FW- nor SVN-bound; return it unmodified.
    return handle;
}

//*** GetAdditionalSecret()
// Retrieve the additional secret for the given hierarchy modifier, along with the
// label that should be used when mixing the secret into a KDF. If the hierarchy
// needs no additional secret, secret_buffer's size is set to zero and secret_label
// is set to NULL.
```

```
//
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM failed
//                              to derive the Firmware SVN Secret for the requested
//                              SVN.
static TPM_RC GetAdditionalSecret(const HIERARCHY_MODIFIER* modifier,       // IN
                                  TPM2B_SEED*               secret_buffer,  // OUT
                                  const TPM2B**             secret_label    // OUT
)
{
    switch(modifier->type)
    {
        case HM_FW_LIMITED:
        {
#if FW_LIMITED_SUPPORT
            if(_plat__GetTpmFirmwareSecret(sizeof(secret_buffer->t.buffer),
                                           secret_buffer->t.buffer,
                                           &secret_buffer->t.size)
               != 0)
            {
                return TPM_RC_FW_LIMITED;
            }

            *secret_label = HIERARCHY_FW_SECRET_LABEL;
            break;
#else
            return TPM_RC_FW_LIMITED;
#endif  // FW_LIMITED_SUPPORT
        }
        case HM_SVN_LIMITED:
        {
#if SVN_LIMITED_SUPPORT
            if(_plat__GetTpmFirmwareSvnSecret(modifier->min_svn,
                                              sizeof(secret_buffer->t.buffer),
                                              secret_buffer->t.buffer,
                                              &secret_buffer->t.size)
               != 0)
            {
                return TPM_RC_SVN_LIMITED;
            }

            *secret_label = HIERARCHY_SVN_SECRET_LABEL;
            break;
#else
            return TPM_RC_SVN_LIMITED;
#endif  // SVN_LIMITED_SUPPORT
        }
        case HM_NONE:
        default:
        {
            secret_buffer->t.size = 0;
            *secret_label         = NULL;
            break;
        }
    }

    return TPM_RC_SUCCESS;
}

//***MixAdditionalSecret()
// This function obtains the additional secret for the hierarchy and
// mixes it into the base secret. The output buffer must have the same
```

```
// capacity as the base secret. The output buffer's size is set to the
// base secret size. If no additional secret is needed, the base secret
// is copied to the output buffer.
//
//  Return Type: TPM_RC
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects or the TPM failed
//                              to derive the Firmware Secret.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the TPM failed
//                              to derive the Firmware SVN Secret for the requested
//                              SVN.
static TPM_RC MixAdditionalSecret(const HIERARCHY_MODIFIER* modifier,           // IN
                                  const TPM2B*              base_secret_label,  // IN
                                  const TPM2B*              base_secret,        // IN
                                  TPM2B*                    output_secret       // OUT
)
{
    TPM_RC       result = TPM_RC_SUCCESS;
    TPM2B_SEED   additional_secret;
    const TPM2B* additional_secret_label = NULL;

    result =
        GetAdditionalSecret(modifier, &additional_secret, &additional_secret_label);
    if(result != TPM_RC_SUCCESS)
        return result;

    output_secret->size = base_secret->size;

    if(additional_secret.b.size == 0)
    {
        memcpy(output_secret->buffer, base_secret->buffer, base_secret->size);
    }
    else
    {
        CryptKDFa(CONTEXT_INTEGRITY_HASH_ALG,
                  base_secret,
                  base_secret_label,
                  &additional_secret.b,
                  additional_secret_label,
                  base_secret->size * 8,
                  output_secret->buffer,
                  NULL,
                  FALSE);
    }

    MemorySet(additional_secret.b.buffer, 0, additional_secret.b.size);

    return TPM_RC_SUCCESS;
}

//*** HierarchyGetProof()
// This function derives the proof value associated with a hierarchy. It returns a
// buffer containing the proof value.
TPM_RC HierarchyGetProof(TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy constant
                         TPM2B_PROOF*      proof       // OUT: proof buffer
)
{
    TPM2B_PROOF*       base_proof = NULL;
    HIERARCHY_MODIFIER modifier;

    switch(DecomposeHandle(hierarchy, &modifier))
    {
        case TPM_RH_PLATFORM:
            // phProof for TPM_RH_PLATFORM
            base_proof = &gp.phProof;
```

```
            break;
        case TPM_RH_ENDORSEMENT:
            // ehProof for TPM_RH_ENDORSEMENT
            base_proof = &gp.ehProof;
            break;
        case TPM_RH_OWNER:
            // shProof for TPM_RH_OWNER
            base_proof = &gp.shProof;
            break;
        default:
            // nullProof for TPM_RH_NULL or anything else
            base_proof = &gr.nullProof;
            break;
    }

    return MixAdditionalSecret(
        &modifier, HIERARCHY_PROOF_SECRET_LABEL, &base_proof->b, &proof->b);
}

//*** HierarchyGetPrimarySeed()
// This function derives the primary seed of a hierarchy.
TPM_RC HierarchyGetPrimarySeed(TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy
                               TPM2B_SEED*       seed        // OUT: seed buffer
)
{
    TPM2B_SEED*        base_seed = NULL;
    HIERARCHY_MODIFIER modifier;

    switch(DecomposeHandle(hierarchy, &modifier))
    {
        case TPM_RH_PLATFORM:
            base_seed = &gp.PPSeed;
            break;
        case TPM_RH_OWNER:
            base_seed = &gp.SPSeed;
            break;
        case TPM_RH_ENDORSEMENT:
            base_seed = &gp.EPSeed;
            break;
        default:
            base_seed = &gr.nullSeed;
            break;
    }

    return MixAdditionalSecret(
        &modifier, HIERARCHY_SEED_SECRET_LABEL, &base_seed->b, &seed->b);
}

//*** ValidateHierarchy()
// This function ensures a given hierarchy is valid and enabled.
//  Return Type: TPM_RC
//      TPM_RC_HIERARCHY        Hierarchy is disabled
//      TPM_RC_FW_LIMITED       The requested hierarchy is FW-limited, but the TPM
//                              does not support FW-limited objects.
//      TPM_RC_SVN_LIMITED      The requested hierarchy is SVN-limited, but the TPM
//                              does not support SVN-limited objects or the given SVN
//                              is greater than the TPM's current SVN.
//      TPM_RC_VALUE            Hierarchy is not valid
TPM_RC ValidateHierarchy(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy
)
{
    BOOL               enabled;
    HIERARCHY_MODIFIER modifier;

    hierarchy = DecomposeHandle(hierarchy, &modifier);
```

```
    // Modifier-specific checks.
    switch(modifier.type)
    {
        case HM_NONE:
            break;
        case HM_FW_LIMITED:
        {
#if FW_LIMITED_SUPPORT
            break;
#else
            return TPM_RC_FW_LIMITED;
#endif  // FW_LIMITED_SUPPORT
        }
        case HM_SVN_LIMITED:
        {
#if SVN_LIMITED_SUPPORT
            // SVN-limited hierarchies are only enabled for SVNs less than or
            // equal to the current firmware's SVN.
            if(modifier.min_svn > _plat__GetTpmFirmwareSvn())
            {
                return TPM_RC_SVN_LIMITED;
            }
            break;
#else
            return TPM_RC_SVN_LIMITED;
#endif  // SVN_LIMITED_SUPPORT
        }
    }

    switch(hierarchy)
    {
        case TPM_RH_PLATFORM:
            enabled = g_phEnable;
            break;
        case TPM_RH_OWNER:
            enabled = gc.shEnable;
            break;
        case TPM_RH_ENDORSEMENT:
            enabled = gc.ehEnable;
            break;
        case TPM_RH_NULL:
            enabled = TRUE;
            break;
        default:
            return TPM_RC_VALUE;
    }

    return enabled ? TPM_RC_SUCCESS : TPM_RC_HIERARCHY;
}

//*** HierarchyIsEnabled()
// This function checks to see if a hierarchy is enabled.
// NOTE: The TPM_RH_NULL hierarchy is always enabled.
//  Return Type: BOOL
//      TRUE(1)         hierarchy is enabled
//      FALSE(0)        hierarchy is disabled
BOOL HierarchyIsEnabled(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy
)
{
    return ValidateHierarchy(hierarchy) == TPM_RC_SUCCESS;
}

//*** HierarchyNormalizeHandle
// This function accepts a handle that may or may not be FW- or SVN-bound,
// and returns the base hierarchy to which the handle refers.
TPMI_RH_HIERARCHY HierarchyNormalizeHandle(TPMI_RH_HIERARCHY handle  // IN: handle
```

```
)
{
    HIERARCHY_MODIFIER unused_modifier;

    return DecomposeHandle(handle, &unused_modifier);
}

//*** HierarchyIsFirmwareLimited
// This function accepts a hierarchy handle and returns whether it is firmware-
// limited.
BOOL HierarchyIsFirmwareLimited(TPMI_RH_HIERARCHY handle  // IN
)
{
    HIERARCHY_MODIFIER modifier;

    DecomposeHandle(handle, &modifier);

    return modifier.type == HM_FW_LIMITED;
}
//*** HierarchyIsSvnLimited
// This function accepts a hierarchy handle and returns whether it is SVN-
// limited.
BOOL HierarchyIsSvnLimited(TPMI_RH_HIERARCHY handle  // IN
)
{
    HIERARCHY_MODIFIER modifier;

    DecomposeHandle(handle, &modifier);

    return modifier.type == HM_SVN_LIMITED;
}
```

7.173 /tpm/src/subsystem/NvDynamic.c

```
//** Introduction

// The NV memory is divided into two areas: dynamic space for user defined NV
// indexes and evict objects, and reserved space for TPM persistent and state save
// data.
//
// The entries in dynamic space are a linked list of entries. Each entry has, as its
// first field, a size. If the size field is zero, it marks the end of the
// list.
//
// An Index allocation will contain an NV_INDEX structure. If the Index does not
// have the orderly attribute, the NV_INDEX is followed immediately by the NV data.
//
// An evict object entry contains a handle followed by an OBJECT structure. This
// results in both the Index and Evict Object having an identifying handle as the
// first field following the size field.
//
// When an Index has the orderly attribute, the data is kept in RAM. This RAM is
// saved to backing store in NV memory on any orderly shutdown. The entries in
// orderly memory are also a linked list using a size field as the first entry.
//
// The attributes of an orderly index are maintained in RAM memory in order to
// reduce the number of NV writes needed for orderly data. When an orderly index
// is created, an entry is made in the dynamic NV memory space that holds the Index
// authorizations (authPolicy and authValue) and the size of the data. This entry is
// only modified if the authValue  of the index is changed. The more volatile data
// of the index is kept in RAM. When an orderly Index is created or deleted, the
// RAM data is copied to NV backing store so that the image in the backing store
// matches the layout of RAM. In normal operation. The RAM data is also copied on
// any orderly shutdown. In normal operation, the only other reason for writing
// to the backing store for RAM is when a counter is first written (TPMA_NV_WRITTEN
```

```
// changes from CLEAR to SET) or when a counter ""rolls over"".
//
// Static space contains items that are individually modifiable. The values are in
// the 'gp' PERSISTENT_DATA structure in RAM and mapped to locations in NV.
//

//** Includes, Defines and Data Definitions
#define NV_C
#include "Tpm.h"
#include "Marshal.h"

//** Local Functions

//*** NvNext()
//  This function provides a method to traverse every data entry in NV dynamic
//  area.
//
//  To begin with, parameter 'iter' should be initialized to NV_REF_INIT
//  indicating the first element.  Every time this function is called, the
//  value in 'iter' would be adjusted pointing to the next element in
//  traversal.  If there is no next element, 'iter' value would be 0.
//  This function returns the address of the 'data entry' pointed by the
//  'iter'.  If there are no more elements in the set, a 0 value is returned
//  indicating the end of traversal.
//
static NV_REF NvNext(NV_REF*     iter,   // IN/OUT: the list iterator
                     TPM_HANDLE* handle  // OUT: the handle of the next item.
)
{
    NV_REF          currentAddr;
    NV_ENTRY_HEADER header;
    //
    // If iterator is at the beginning of list
    if(*iter == NV_REF_INIT)
    {
        // Initialize iterator
        *iter = NV_USER_DYNAMIC;
    }
    // Step over the size field and point to the handle
    currentAddr = *iter + sizeof(UINT32);

    // read the header of the next entry
    NvRead(&header, *iter, sizeof(NV_ENTRY_HEADER));

    // if the size field is zero, then we have hit the end of the list
    if(header.size == 0)
        // leave the *iter pointing at the end of the list
        return 0;
    // advance the header by the size of the entry
    *iter += header.size;

    if(handle != NULL)
        *handle = header.handle;
    return currentAddr;
}

//*** NvNextByType()
// This function returns a reference to the next NV entry of the desired type
//  Return Type: NV_REF
//      0               end of list
//      != 0            the next entry of the indicated type
static NV_REF NvNextByType(
    TPM_HANDLE* handle,  // OUT: the handle of the found type or 0
    NV_REF*     iter,    // IN: the iterator
    TPM_HT      type     // IN: the handle type to look for
)
```

```
{
    NV_REF     addr;
    TPM_HANDLE nvHandle = 0;
    //
    while((addr = NvNext(iter, &nvHandle)) != 0)
    {
        // addr: the address of the location containing the handle of the value
        // iter: the next location.
        if(HandleGetType(nvHandle) == type)
            break;
    }
    if(handle != NULL)
        *handle = nvHandle;
    return addr;
}

//*** NvNextIndex()
// This function returns the reference to the next NV Index entry.  A value
// of 0 indicates the end of the list.
//  Return Type: NV_REF
//      0               end of list
//      != 0            the next reference
#define NvNextIndex(handle, iter) NvNextByType(handle, iter, TPM_HT_NV_INDEX)

//*** NvNextEvict()
// This function returns the offset in NV of the next evict object entry.  A value
// of 0 indicates the end of the list.
#define NvNextEvict(handle, iter) NvNextByType(handle, iter, TPM_HT_PERSISTENT)

//*** NvGetEnd()
// Function to find the end of the NV dynamic data list
static NV_REF NvGetEnd(void)
{
    NV_REF iter = NV_REF_INIT;
    NV_REF currentAddr;
    //
    // Scan until the next address is 0
    while((currentAddr = NvNext(&iter, NULL)) != 0)
        ;
    return iter;
}

//*** NvGetFreeBytes
// This function returns the number of free octets in NV space.
static UINT32 NvGetFreeBytes(void)
{
    // This does not have an overflow issue because NvGetEnd() cannot return a value
    // that is larger than s_evictNvEnd. This is because there is always a 'stop'
    // word in the NV memory that terminates the search for the end before the
    // value can go past s_evictNvEnd.
    return s_evictNvEnd - NvGetEnd();
}

//*** NvTestSpace()
// This function will test if there is enough space to add a new entity.
//  Return Type: BOOL
//      TRUE(1)         space available
//      FALSE(0)        no enough space
static BOOL NvTestSpace(UINT32 size,      // IN: size of the entity to be added
                        BOOL   isIndex,   // IN: TRUE if the entity is an index
                        BOOL   isCounter  // IN: TRUE if the index is a counter
)
{
    UINT32 remainBytes = NvGetFreeBytes();
    UINT32 reserved    = sizeof(UINT32)  // size of the forward pointer
                       + sizeof(NV_LIST_TERMINATOR);
```

```
//
// Do a compile time sanity check on the setting for NV_MEMORY_SIZE
#if NV_MEMORY_SIZE < 1024
#  error "NV_MEMORY_SIZE probably isn't large enough"
#endif

    // For NV Index, need to make sure that we do not allocate an Index if this
    // would mean that the TPM cannot allocate the minimum number of evict
    // objects.
    if(isIndex)
    {
        // Get the number of persistent objects allocated
        UINT32 persistentNum = NvCapGetPersistentNumber();

        // If we have not allocated the requisite number of evict objects, then we
        // need to reserve space for them.
        // NOTE: some of this is not written as simply as it might seem because
        // the values are all unsigned and subtracting needs to be done carefully
        // so that an underflow doesn't cause problems.
        if(persistentNum < MIN_EVICT_OBJECTS)
            reserved += (MIN_EVICT_OBJECTS - persistentNum) * NV_EVICT_OBJECT_SIZE;
    }
    // If this is not an index or is not a counter, reserve space for the
    // required number of counter indexes
    if(!isIndex || !isCounter)
    {
        // Get the number of counters
        UINT32 counterNum = NvCapGetCounterNumber();

        // If the required number of counters have not been allocated, reserved
        // space for the extra needed counters
        if(counterNum < MIN_COUNTER_INDICES)
            reserved += (MIN_COUNTER_INDICES - counterNum) * NV_INDEX_COUNTER_SIZE;
    }
    // Check that the requested allocation will fit after making sure that there
    // will be no chance of overflow
    return ((reserved < remainBytes) && (size <= remainBytes)
            && (size + reserved <= remainBytes));
}

//*** NvWriteNvListEnd()
// Function to write the list terminator.
NV_REF
NvWriteNvListEnd(NV_REF end)
{
    // Marker is initialized with zeros
    BYTE   listEndMarker[sizeof(NV_LIST_TERMINATOR)] = {0};
    UINT64 maxCount                                  = NvReadMaxCount();
    //
    // This is a constant check that can be resolved at compile time.
    MUST_BE(sizeof(UINT64) <= sizeof(NV_LIST_TERMINATOR) - sizeof(UINT32));

    // Copy the maxCount value to the marker buffer
    MemoryCopy(&listEndMarker[sizeof(UINT32)], &maxCount, sizeof(UINT64));
    pAssert(end + sizeof(NV_LIST_TERMINATOR) <= s_evictNvEnd);

    // Write it to memory
    NvWrite(end, sizeof(NV_LIST_TERMINATOR), &listEndMarker);
    return end + sizeof(NV_LIST_TERMINATOR);
}

//*** NvAdd()
// This function adds a new entity to NV.
//
// This function requires that there is enough space to add a new entity (i.e.,
// that NvTestSpace() has been called and the available space is at least as
```

```
// large as the required space).
//
// The 'totalSize' will be the size of 'entity'. If a handle is added, this
// function will increase the size accordingly.
static TPM_RC NvAdd(UINT32 totalSize,  // IN: total size needed for this entity For
                                       //     evict object, totalSize is the same as
                                       //     bufferSize.  For NV Index, totalSize is
                                       //     bufferSize plus index data size
                    UINT32     bufferSize,  // IN: size of initial buffer
                    TPM_HANDLE handle,      // IN: optional handle
                    BYTE*      entity       // IN: initial buffer
)
{
    NV_REF newAddr;  // IN: where the new entity will start
    NV_REF nextAddr;
    //
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Get the end of data list
    newAddr = NvGetEnd();

    // Step over the forward pointer
    nextAddr = newAddr + sizeof(UINT32);

    // Optionally write the handle. For indexes, the handle is TPM_RH_UNASSIGNED
    // so that the handle in the nvIndex is used instead of writing this value
    if(handle != TPM_RH_UNASSIGNED)
    {
        NvWrite((UINT32)nextAddr, sizeof(TPM_HANDLE), &handle);
        nextAddr += sizeof(TPM_HANDLE);
    }
    // Write entity data
    NvWrite((UINT32)nextAddr, bufferSize, entity);

    // Advance the pointer by the amount of the total
    nextAddr += totalSize;

    // Finish by writing the link value

    // Write the next offset (relative addressing)
    totalSize = nextAddr - newAddr;

    // Write link value
    NvWrite((UINT32)newAddr, sizeof(UINT32), &totalSize);

    // Write the list terminator
    NvWriteNvListEnd(nextAddr);

    return TPM_RC_SUCCESS;
}

//*** NvDelete()
// This function is used to delete an NV Index or persistent object from NV memory.
static TPM_RC NvDelete(NV_REF entityRef  // IN: reference to entity to be deleted
)
{
    UINT32 entrySize;
    // adjust entityAddr to back up and point to the forward pointer
    NV_REF entryRef = entityRef - sizeof(UINT32);
    NV_REF endRef   = NvGetEnd();
    NV_REF nextAddr;  // address of the next entry
                      //
    RETURN_IF_NV_IS_NOT_AVAILABLE;

    // Get the offset of the next entry. That is, back up and point to the size
    // field of the entry
```

```
    NvRead(&entrySize, entryRef, sizeof(UINT32));

    // The next entry after the one being deleted is at a relative offset
    // from the current entry
    nextAddr = entryRef + entrySize;

    // If this is not the last entry, move everything up
    if(nextAddr < endRef)
    {
        pAssert(nextAddr > entryRef);
        _plat__NvMemoryMove(nextAddr, entryRef, (endRef - nextAddr));
    }
    // The end of the used space is now moved up by the amount of space we just
    // reclaimed
    endRef -= entrySize;

    // Write the end marker, and make the new end equal to the first byte after
    // the just added end value. This will automatically update the NV value for
    // maxCounter.
    // NOTE: This is the call that sets flag to cause NV to be updated
    endRef = NvWriteNvListEnd(endRef);

    // Clear the reclaimed memory
    _plat__NvMemoryClear(endRef, entrySize);

    return TPM_RC_SUCCESS;
}

//**********************************************
//** RAM-based NV Index Data Access Functions
//**********************************************
//*** Introduction
// The data layout in ram buffer is {size of(NV_handle + attributes + data
// NV_handle, attributes, data}
// for each NV Index data stored in RAM.
//
// NV storage associated with orderly data is updated when a NV Index is added
// but NOT when the data or attributes are changed. Orderly data is only updated
// to NV on an orderly shutdown (TPM2_Shutdown())

//*** NvRamNext()
// This function is used to iterate trough the list of Ram Index values. *iter needs
// to be initialized by calling
static NV_RAM_REF NvRamNext(NV_RAM_REF* iter,   // IN/OUT: the list iterator
                            TPM_HANDLE* handle  // OUT: the handle of the next item.
)
{
    NV_RAM_REF    currentAddr;
    NV_RAM_HEADER header;
    //
    // If iterator is at the beginning of list
    if(*iter == NV_RAM_REF_INIT)
    {
        // Initialize iterator
        *iter = &s_indexOrderlyRam[0];
    }
    // if we are going to return what the iter is currently pointing to...
    currentAddr = *iter;

    // If iterator reaches the end of NV space, then don't advance and return
    // that we are at the end of the list. The end of the list occurs when
    // we don't have space for a size and a handle
    if(currentAddr + sizeof(NV_RAM_HEADER) > RAM_ORDERLY_END)
        return NULL;
    // read the header of the next entry
    MemoryCopy(&header, currentAddr, sizeof(NV_RAM_HEADER));
```

```
    // if the size field is zero, then we have hit the end of the list
    if(header.size == 0)
        // leave the *iter pointing at the end of the list
        return NULL;
    // advance the header by the size of the entry
    *iter = currentAddr + header.size;

    //    pAssert(*iter <= RAM_ORDERLY_END);
    if(handle != NULL)
        *handle = header.handle;
    return currentAddr;
}

//*** NvRamGetEnd()
// This routine performs the same function as NvGetEnd() but for the RAM data.
static NV_RAM_REF NvRamGetEnd(void)
{
    NV_RAM_REF iter = NV_RAM_REF_INIT;
    NV_RAM_REF currentAddr;
    //
    // Scan until the next address is 0
    while((currentAddr = NvRamNext(&iter, NULL)) != 0)
        ;
    return iter;
}

//*** NvRamTestSpaceIndex()
// This function indicates if there is enough RAM space to add a data for a
// new NV Index.
//  Return Type: BOOL
//      TRUE(1)         space available
//      FALSE(0)        no enough space
static BOOL NvRamTestSpaceIndex(
    UINT32 size  // IN: size of the data to be added to RAM
)
{
    UINT32 remaining = (UINT32)(RAM_ORDERLY_END - NvRamGetEnd());
    UINT32 needed    = sizeof(NV_RAM_HEADER) + size;
    //
    // NvRamGetEnd points to the next available byte.
    return remaining >= needed;
}

//*** NvRamGetIndex()
// This function returns the offset of NV data in the RAM buffer
//
// This function requires that NV Index is in RAM. That is, the
// index must be known to exist.
static NV_RAM_REF NvRamGetIndex(TPMI_RH_NV_INDEX handle  // IN: NV handle
)
{
    NV_RAM_REF iter = NV_RAM_REF_INIT;
    NV_RAM_REF currentAddr;
    TPM_HANDLE foundHandle;
    //
    while((currentAddr = NvRamNext(&iter, &foundHandle)) != 0)
    {
        if(handle == foundHandle)
            break;
    }
    return currentAddr;
}

//*** NvUpdateIndexOrderlyData()
// This function is used to cause an update of the orderly data to the NV backing
```

```
// store.
void NvUpdateIndexOrderlyData(void)
{
    // Write reserved RAM space to NV
    NvWrite(NV_INDEX_RAM_DATA, sizeof(s_indexOrderlyRam), s_indexOrderlyRam);
}

//*** NvAddRAM()
// This function adds a new data area to RAM.
//
// This function requires that enough free RAM space is available to add
// the new data.
//
// This function should be called after the NV Index space has been updated
// and the index removed. This insures that NV is available so that checking
// for NV availability is not required during this function.
static void NvAddRAM(TPMS_NV_PUBLIC* index  // IN: the index descriptor
)
{
    NV_RAM_HEADER header;
    NV_RAM_REF    end = NvRamGetEnd();
    //
    header.size   = sizeof(NV_RAM_HEADER) + index->dataSize;
    header.handle = index->nvIndex;
    MemoryCopy(&header.attributes, &index->attributes, sizeof(TPMA_NV));

    pAssert(ORDERLY_RAM_ADDRESS_OK(end, header.size));

    // Copy the header to the memory
    MemoryCopy(end, &header, sizeof(NV_RAM_HEADER));

    // Clear the data area (just in case)
    MemorySet(end + sizeof(NV_RAM_HEADER), 0, index->dataSize);

    // Step over this new entry
    end += header.size;

    // If the end marker will fit, add it
    if(end + sizeof(UINT32) < RAM_ORDERLY_END)
        MemorySet(end, 0, sizeof(UINT32));
    // Write reserved RAM space to NV to reflect the newly added NV Index
    SET_NV_UPDATE(UT_ORDERLY);

    return;
}

//*** NvDeleteRAM()
// This function is used to delete a RAM-backed NV Index data area.
// The space used by the entry are overwritten by the contents of the
// Index data that comes after (the data is moved up to fill the hole left
// by removing this index. The reclaimed space is cleared to zeros.
// This function assumes the data of NV Index exists in RAM.
//
// This function should be called after the NV Index space has been updated
// and the index removed. This insures that NV is available so that checking
// for NV availability is not required during this function.
static void NvDeleteRAM(TPMI_RH_NV_INDEX handle  // IN: NV handle
)
{
    NV_RAM_REF nodeAddress;
    NV_RAM_REF nextNode;
    UINT32     size;
    NV_RAM_REF lastUsed = NvRamGetEnd();
    //
    nodeAddress = NvRamGetIndex(handle);
```

```
    pAssert(nodeAddress != 0);

    // Get node size
    MemoryCopy(&size, nodeAddress, sizeof(size));

    // Get the offset of next node
    nextNode = nodeAddress + size;

    // Copy the data
    MemoryCopy(nodeAddress, nextNode, (int)(lastUsed - nextNode));

    // Clear out the reclaimed space
    MemorySet(lastUsed - size, 0, size);

    // Write reserved RAM space to NV to reflect the newly delete NV Index
    SET_NV_UPDATE(UT_ORDERLY);

    return;
}

//*** NvReadIndex()
// This function is used to read the NV Index NV_INDEX. This is used so that the
// index information can be compressed and only this function would be needed
// to decompress it. Mostly, compression would only be able to save the space
// needed by the policy.
void NvReadNvIndexInfo(NV_REF    ref,     // IN: points to NV where index is located
                       NV_INDEX* nvIndex  // OUT: place to receive index data
)
{
    pAssert(nvIndex != NULL);
    NvRead(nvIndex, ref, sizeof(NV_INDEX));
    return;
}

//*** NvReadObject()
// This function is used to read a persistent object. This is used so that the
// object information can be compressed and only this function would be needed
// to uncompress it.
void NvReadObject(NV_REF  ref,    // IN: points to NV where index is located
                  OBJECT* object  // OUT: place to receive the object data
)
{
    NvRead(object, (ref + sizeof(TPM_HANDLE)), sizeof(OBJECT));
    return;
}

//*** NvFindEvict()
// This function will return the NV offset of an evict object
//  Return Type: UINT32
//      0               evict object not found
//      != 0            offset of evict object
static NV_REF NvFindEvict(TPM_HANDLE nvHandle, OBJECT* object)
{
    NV_REF found = NvFindHandle(nvHandle);
    //
    // If we found the handle and the request included an object pointer, fill it in
    if(found != 0 && object != NULL)
        NvReadObject(found, object);
    return found;
}

//*** NvIndexIsDefined()
// See if an index is already defined
BOOL NvIndexIsDefined(TPM_HANDLE nvHandle  // IN: Index to look for
)
{
```

```
    return (NvFindHandle(nvHandle) != 0);
}

//*** NvConditionallyWrite()
// Function to check if the data to be written has changed
// and write it if it has
//  Return Type: TPM_RC
//      TPM_RC_NV_RATE          NV is unavailable because of rate limit
//      TPM_RC_NV_UNAVAILABLE   NV is inaccessible
static TPM_RC NvConditionallyWrite(NV_REF entryAddr,  // IN: stating address
                                   UINT32 size,       // IN: size of the data to write
                                   void*  data        // IN: the data to write
)
{
    // If the index data is actually changed, then a write to NV is required
    int isDifferent = _plat__NvGetChangedStatus(entryAddr, size, data);
    if(isDifferent == NV_INVALID_LOCATION)
    {
        // invalid request, we should be in failure mode by now.
        return TPM_RC_FAILURE;
    }
    else if(isDifferent == NV_HAS_CHANGED)
    {
        // Write the data if NV is available
        if(g_NvStatus == TPM_RC_SUCCESS)
        {
            NvWrite(entryAddr, size, data);
        }
        return g_NvStatus;
    }
    else if(isDifferent == NV_IS_SAME)
    {
        return TPM_RC_SUCCESS;
    }
    // the platform gave us an invalid response.
    FAIL_RC(FATAL_ERROR_PLATFORM);
}

//*** NvReadNvIndexAttributes()
// This function returns the attributes of an NV Index.
static TPMA_NV NvReadNvIndexAttributes(NV_REF locator  // IN: reference to an NV index
)
{
    TPMA_NV attributes;
    //
    NvRead(&attributes,
           locator + offsetof(NV_INDEX, publicArea.attributes),
           sizeof(TPMA_NV));
    return attributes;
}

//*** NvReadRamIndexAttributes()
// This function returns the attributes from the RAM header structure. This function
// is used to deal with the fact that the header structure is only byte aligned.
static TPMA_NV NvReadRamIndexAttributes(
    NV_RAM_REF ref  // IN: pointer to a NV_RAM_HEADER
)
{
    TPMA_NV attributes;
    //
    MemoryCopy(
        &attributes, ref + offsetof(NV_RAM_HEADER, attributes), sizeof(TPMA_NV));
    return attributes;
}

//*** NvWriteNvIndexAttributes()
```

```
// This function is used to write just the attributes of an index to NV.
//  Return type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting so retry
//      TPM_RC_NV_UNAVAILABLE   NV is not available
static TPM_RC NvWriteNvIndexAttributes(NV_REF  locator,  // IN: location of the index
                                       TPMA_NV attributes  // IN: attributes to write
)
{
    return NvConditionallyWrite(locator + offsetof(NV_INDEX, publicArea.attributes),
                                sizeof(TPMA_NV),
                                &attributes);
}

//*** NvWriteRamIndexAttributes()
// This function is used to write the index attributes into an unaligned structure
static void NvWriteRamIndexAttributes(
    NV_RAM_REF ref,         // IN: address of the header
    TPMA_NV    attributes  // IN: the attributes to write
)
{
    MemoryCopy(
        ref + offsetof(NV_RAM_HEADER, attributes), &attributes, sizeof(TPMA_NV));
    return;
}

//**********************************************
//** Externally Accessible Functions
//**********************************************

//*** NvIsPlatformPersistentHandle()
// This function indicates if a handle references a persistent object in the
// range belonging to the platform.
//  Return Type: BOOL
//      TRUE(1)         handle references a platform persistent object
//      FALSE(0)        handle does not reference platform persistent object
BOOL NvIsPlatformPersistentHandle(TPM_HANDLE handle  // IN: handle
)
{
    return (handle >= PLATFORM_PERSISTENT && handle <= PERSISTENT_LAST);
}

//*** NvIsOwnerPersistentHandle()
// This function indicates if a handle references a persistent object in the
// range belonging to the owner.
//  Return Type: BOOL
//      TRUE(1)         handle is owner persistent handle
//      FALSE(0)        handle is not owner persistent handle and may not be
//                      a persistent handle at all
BOOL NvIsOwnerPersistentHandle(TPM_HANDLE handle  // IN: handle
)
{
    return (handle >= PERSISTENT_FIRST && handle < PLATFORM_PERSISTENT);
}

//*** NvIndexIsAccessible()
//
// This function validates that a handle references a defined NV Index and
// that the Index is currently accessible.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE           the handle points to an undefined NV Index
//                              If shEnable is CLEAR, this would include an index
//                              created using ownerAuth. If phEnableNV is CLEAR,
//                              this would include and index created using
//                              platformAuth
//      TPM_RC_NV_READLOCKED    Index is present but locked for reading and command
//                              does not write to the index
```

```
//      TPM_RC_NV_WRITELOCKED   Index is present but locked for writing and command
//                              writes to the index
TPM_RC
NvIndexIsAccessible(TPMI_RH_NV_INDEX handle  // IN: handle
)
{
    NV_INDEX* nvIndex = NvGetIndexInfo(handle, NULL);
    //
    if(nvIndex == NULL)
        // If index is not found, return TPM_RC_HANDLE
        return TPM_RC_HANDLE;
    if(gc.shEnable == FALSE || gc.phEnableNV == FALSE)
    {
        // if shEnable is CLEAR, an ownerCreate NV Index should not be
        // indicated as present
        if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, PLATFORMCREATE))
        {
            if(gc.shEnable == FALSE)
                return TPM_RC_HANDLE;
        }
        // if phEnableNV is CLEAR, a platform created Index should not
        // be visible
        else if(gc.phEnableNV == FALSE)
            return TPM_RC_HANDLE;
    }
#if 0  // Writelock test for debug
    // If the Index is write locked and this is an NV Write operation...
    if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITELOCKED)
       &&  IsWriteOperation(commandIndex))
    {
        // then return a locked indication unless the command is TPM2_NV_WriteLock
        if(GetCommandCode(commandIndex) != TPM_CC_NV_WriteLock)
            return TPM_RC_NV_LOCKED;
        return TPM_RC_SUCCESS;
    }
#endif
#if 0  // Readlock Test for debug
    // If the Index is read locked and this is an NV Read operation...
    if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, READLOCKED)
       && IsReadOperation(commandIndex))
    {
        // then return a locked indication unless the command is TPM2_NV_ReadLock
        if(GetCommandCode(commandIndex) != TPM_CC_NV_ReadLock)
            return TPM_RC_NV_LOCKED;
    }
#endif
    // NV Index is accessible
    return TPM_RC_SUCCESS;
}

//*** NvGetEvictObject()
// This function is used to dereference an evict object handle and get a pointer
// to the object.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE           the handle does not point to an existing
//                              persistent object
TPM_RC
NvGetEvictObject(TPM_HANDLE handle,  // IN: handle
                 OBJECT*    object   // OUT: object data
)
{
    NV_REF entityAddr;  // offset points to the entity
                        //
    // Find the address of evict object and copy to object
    entityAddr = NvFindEvict(handle, object);
```

```
    // whether there is an error or not, make sure that the evict
    // status of the object is set so that the slot will get freed on exit
    // Must do this after NvFindEvict loads the object
    object->attributes.evict = SET;

    // If handle is not found, return an error
    if(entityAddr == 0)
        return TPM_RC_HANDLE;
    return TPM_RC_SUCCESS;
}

//*** NvIndexCacheInit()
// Function to initialize the Index cache
void NvIndexCacheInit(void)
{
    s_cachedNvRef                       = NV_REF_INIT;
    s_cachedNvRamRef                    = NV_RAM_REF_INIT;
    s_cachedNvIndex.publicArea.nvIndex = TPM_RH_UNASSIGNED;
    return;
}

//*** NvGetIndexData()
// This function is used to access the data in an NV Index. The data is returned
// as a byte sequence.
//
// This function requires that the NV Index be defined, and that the
// required data is within the data range.  It also requires that TPMA_NV_WRITTEN
// of the Index is SET.
void NvGetIndexData(NV_INDEX* nvIndex,  // IN: the in RAM index descriptor
                    NV_REF    locator,  // IN: where the data is located
                    UINT32    offset,   // IN: offset of NV data
                    UINT16    size,     // IN: number of octets of NV data to read
                    void*     data      // OUT: data buffer
)
{
    TPMA_NV nvAttributes;
    //
    pAssert(nvIndex != NULL);

    nvAttributes = nvIndex->publicArea.attributes;

    pAssert(IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITTEN));

    if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, ORDERLY))
    {
        // Get data from RAM buffer
        NV_RAM_REF ramAddr = NvRamGetIndex(nvIndex->publicArea.nvIndex);
        pAssert(ramAddr != 0
                && (size <= ((NV_RAM_HEADER*)ramAddr)->size - sizeof(NV_RAM_HEADER)
                                - offset));
        MemoryCopy(data, ramAddr + sizeof(NV_RAM_HEADER) + offset, size);
    }
    else
    {
        // Validate that read falls within range of the index
        pAssert(offset <= nvIndex->publicArea.dataSize
                && size <= (nvIndex->publicArea.dataSize - offset));
        NvRead(data, locator + sizeof(NV_INDEX) + offset, size);
    }
    return;
}

//*** NvHashIndexData()
// This function adds Index data to a hash. It does this in parts to avoid large stack
// buffers.
void NvHashIndexData(HASH_STATE* hashState,  // IN: Initialized hash state
```

```
                    NV_INDEX*   nvIndex,    // IN: Index
                    NV_REF      locator,    // IN: where the data is located
                    UINT32      offset,     // IN: starting offset
                    UINT16      size        // IN: amount to hash
)
{
#define BUFFER_SIZE 64
    BYTE buffer[BUFFER_SIZE];
    if(offset > nvIndex->publicArea.dataSize)
        return;
    // Make sure that we don't try to read off the end.
    if((offset + size) > nvIndex->publicArea.dataSize)
        size = nvIndex->publicArea.dataSize - (UINT16)offset;
#if BUFFER_SIZE >= MAX_NV_INDEX_SIZE
    NvGetIndexData(nvIndex, locator, offset, size, buffer);
    CryptDigestUpdate(hashState, size, buffer);
#else
    {
        INT16  i;
        UINT16 readSize;
        //
        for(i = size; i > 0; offset += readSize, i -= readSize)
        {
            readSize = (i < BUFFER_SIZE) ? i : BUFFER_SIZE;
            NvGetIndexData(nvIndex, locator, offset, readSize, buffer);
            CryptDigestUpdate(hashState, readSize, buffer);
        }
    }
#endif  // BUFFER_SIZE >= MAX_NV_INDEX_SIZE
#undef BUFFER_SIZE
}

//*** NvGetUINT64Data()
// Get data in integer format of a bit or counter NV Index.
//
// This function requires that the NV Index is defined and that the NV Index
// previously has been written.
UINT64
NvGetUINT64Data(NV_INDEX* nvIndex,  // IN: the in RAM index descriptor
                NV_REF    locator   // IN: where index exists in NV
)
{
    UINT64 intVal;
    //
    // Read the value and convert it to internal format
    NvGetIndexData(nvIndex, locator, 0, 8, &intVal);
    return BYTE_ARRAY_TO_UINT64(((BYTE*)&intVal));
}

//*** NvWriteIndexAttributes()
// This function is used to write just the attributes of an index.
//  Return type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting so retry
//      TPM_RC_NV_UNAVAILABLE   NV is not available
TPM_RC
NvWriteIndexAttributes(TPM_HANDLE handle,
                       NV_REF     locator,    // IN: location of the index
                       TPMA_NV    attributes  // IN: attributes to write
)
{
    TPM_RC result;
    //
    if(IS_ATTRIBUTE(attributes, TPMA_NV, ORDERLY))
    {
        NV_RAM_REF ram = NvRamGetIndex(handle);
        NvWriteRamIndexAttributes(ram, attributes);
```

```
        result = TPM_RC_SUCCESS;
    }
    else
    {
        result = NvWriteNvIndexAttributes(locator, attributes);
    }
    return result;
}

//*** NvWriteIndexAuth()
// This function is used to write the authValue of an index. It is used by
// TPM2_NV_ChangeAuth()
//  Return type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting so retry
//      TPM_RC_NV_UNAVAILABLE   NV is not available
TPM_RC
NvWriteIndexAuth(NV_REF      locator,   // IN: location of the index
                 TPM2B_AUTH* authValue  // IN: the authValue to write
)
{
    TPM_RC result;
    //
    // If the locator is pointing to the cached index value...
    if(locator == s_cachedNvRef)
    {
        // copy the authValue to the cached index so it will be there if we
        // look for it. This is a safety thing.
        MemoryCopy2B(&s_cachedNvIndex.authValue.b,
                     &authValue->b,
                     sizeof(s_cachedNvIndex.authValue.t.buffer));
    }
    result = NvConditionallyWrite(locator + offsetof(NV_INDEX, authValue),
                                  sizeof(UINT16) + authValue->t.size,
                                  authValue);
    return result;
}

//*** NvGetIndexInfo()
// This function loads the nvIndex Info into the NV cache and returns a pointer
// to the NV_INDEX. If the returned value is zero, the index was not found.
// The 'locator' parameter, if not NULL, will be set to the offset in NV of the
// Index (the location of the handle of the Index).
//
// This function will set the index cache. If the index is orderly, the attributes
// from RAM are substituted for the attributes in the cached index
NV_INDEX* NvGetIndexInfo(TPM_HANDLE nvHandle,  // IN: the index handle
                         NV_REF*    locator    // OUT: location of the index
)
{
    if(s_cachedNvIndex.publicArea.nvIndex != nvHandle)
    {
        s_cachedNvIndex.publicArea.nvIndex = TPM_RH_UNASSIGNED;
        s_cachedNvRamRef                   = 0;
        s_cachedNvRef                      = NvFindHandle(nvHandle);
        if(s_cachedNvRef == 0)
            return NULL;
        NvReadNvIndexInfo(s_cachedNvRef, &s_cachedNvIndex);
        if(IS_ATTRIBUTE(s_cachedNvIndex.publicArea.attributes, TPMA_NV, ORDERLY))
        {
            s_cachedNvRamRef = NvRamGetIndex(nvHandle);
            s_cachedNvIndex.publicArea.attributes =
                NvReadRamIndexAttributes(s_cachedNvRamRef);
        }
    }
    if(locator != NULL)
        *locator = s_cachedNvRef;
```

```
    return &s_cachedNvIndex;
}

//*** NvWriteIndexData()
// This function is used to write NV index data. It is intended to be used to
// update the data associated with the default index.
//
// This function requires that the NV Index is defined, and the data is
// within the defined data range for the index.
//
// Index data is only written due to a command that modifies the data in a single
// index. There is no case where changes are made to multiple indexes data at the
// same time. Multiple attributes may be change but not multiple index data. This
// is important because we will normally be handling the index for which we have
// the cached pointer values.
//  Return type: TPM_RC
//      TPM_RC_NV_RATE          NV is rate limiting so retry
//      TPM_RC_NV_UNAVAILABLE   NV is not available
TPM_RC
NvWriteIndexData(NV_INDEX* nvIndex,  // IN: the description of the index
                 UINT32    offset,   // IN: offset of NV data
                 UINT32    size,     // IN: size of NV data
                 void*     data      // IN: data buffer
)
{
    TPM_RC result = TPM_RC_SUCCESS;
    //
    pAssert(nvIndex != NULL);
    // Make sure that this is dealing with the 'default' index.
    // Note: it is tempting to change the calling sequence so that the 'default' is
    // presumed.
    pAssert(nvIndex->publicArea.nvIndex == s_cachedNvIndex.publicArea.nvIndex);

    // Validate that write falls within range of the index
    pAssert(offset <= nvIndex->publicArea.dataSize
            && size <= (nvIndex->publicArea.dataSize - offset));

    // Update TPMA_NV_WRITTEN bit if necessary
    if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
    {
        // Update the in memory version of the attributes
        SET_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN);

        // If this is not orderly, then update the NV version of
        // the attributes
        if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, ORDERLY))
        {
            result = NvWriteNvIndexAttributes(s_cachedNvRef,
                                              nvIndex->publicArea.attributes);
            if(result != TPM_RC_SUCCESS)
                return result;
            // If this is a partial write of an ordinary index, clear the whole
            // index.
            if(IsNvOrdinaryIndex(nvIndex->publicArea.attributes)
               && (nvIndex->publicArea.dataSize > size))
                _plat__NvMemoryClear(s_cachedNvRef + sizeof(NV_INDEX),
                                     nvIndex->publicArea.dataSize);
        }
        else
        {
            // This is orderly so update the RAM version
            MemoryCopy(s_cachedNvRamRef + offsetof(NV_RAM_HEADER, attributes),
                       &nvIndex->publicArea.attributes,
                       sizeof(TPMA_NV));
            // If setting WRITTEN for an orderly counter, make sure that the
            // state saved version of the counter is saved
```

```
            if(IsNvCounterIndex(nvIndex->publicArea.attributes))
                SET_NV_UPDATE(UT_ORDERLY);
            // If setting the written attribute on an ordinary index, make sure that
            // the data is all cleared out in case there is a partial write. This
            // is only necessary for ordinary indexes because all of the other types
            // are always written in total.
            else if(IsNvOrdinaryIndex(nvIndex->publicArea.attributes))
                MemorySet(s_cachedNvRamRef + sizeof(NV_RAM_HEADER),
                          0,
                          nvIndex->publicArea.dataSize);
        }
    }
    // If this is orderly data, write it to RAM
    if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, ORDERLY))
    {
        // Note: if this is the first write to a counter, the code above will queue
        // the write to NV of the RAM data in order to update TPMA_NV_WRITTEN. In
        // process of doing that write, it will also write the initial counter value

        // Update RAM
        MemoryCopy(s_cachedNvRamRef + sizeof(NV_RAM_HEADER) + offset, data, size);

        // And indicate that the TPM is no longer orderly
        g_clearOrderly = TRUE;
    }
    else
    {
        // Offset into the index to the first byte of the data to be written to NV
        result = NvConditionallyWrite(
            s_cachedNvRef + sizeof(NV_INDEX) + offset, size, data);
    }
    return result;
}

//*** NvWriteUINT64Data()
// This function to write back a UINT64 value. The various UINT64 values (bits,
// counters, and PINs) are kept in canonical format but manipulate in native
// format. This takes a native format value converts it and saves it back as
// in canonical format.
//
// This function will return the value from NV or RAM depending on the type of the
// index (orderly or not)
//
TPM_RC
NvWriteUINT64Data(NV_INDEX* nvIndex,  // IN: the description of the index
                  UINT64    intValue  // IN: the value to write
)
{
    BYTE bytes[8];
    UINT64_TO_BYTE_ARRAY(intValue, bytes);
    //
    return NvWriteIndexData(nvIndex, 0, 8, &bytes);
}

//*** NvGetNameByIndexHandle()
// This function is used to compute the Name of an NV Index referenced by handle.
//
// The 'name' buffer receives the bytes of the Name and the return value
// is the number of octets in the Name.
//
// This function requires that the NV Index is defined.
TPM2B_NAME* NvGetNameByIndexHandle(
    TPMI_RH_NV_INDEX handle,  // IN: handle of the index
    TPM2B_NAME*      name     // OUT: name of the index
)
{
```

```
    NV_INDEX* nvIndex = NvGetIndexInfo(handle, NULL);
    //
    return NvGetIndexName(nvIndex, name);
}

//*** NvDefineIndex()
// This function is used to assign NV memory to an NV Index.
//
//  Return Type: TPM_RC
//      TPM_RC_NV_SPACE         insufficient NV space
TPM_RC
NvDefineIndex(TPMS_NV_PUBLIC* publicArea,  // IN: A template for an area to create.
              TPM2B_AUTH*     authValue    // IN: The initial authorization value
)
{
    // The buffer to be written to NV memory
    NV_INDEX nvIndex;    // the index data
    UINT16   entrySize;  // size of entry
    TPM_RC   result;
    //
    entrySize = sizeof(NV_INDEX);

    // only allocate data space for indexes that are going to be written to NV.
    // Orderly indexes don't need space.
    if(!IS_ATTRIBUTE(publicArea->attributes, TPMA_NV, ORDERLY))
        entrySize += publicArea->dataSize;
    // Check if we have enough space to create the NV Index
    // In this implementation, the only resource limitation is the available NV
    // space (and possibly RAM space.)  Other implementation may have other
    // limitation on counter or on NV slots
    if(!NvTestSpace(entrySize, TRUE, IsNvCounterIndex(publicArea->attributes)))
        return TPM_RC_NV_SPACE;

    // if the index to be defined is RAM backed, check RAM space availability
    // as well
    if(IS_ATTRIBUTE(publicArea->attributes, TPMA_NV, ORDERLY)
       && !NvRamTestSpaceIndex(publicArea->dataSize))
        return TPM_RC_NV_SPACE;
    // Copy input value to nvBuffer
    nvIndex.publicArea = *publicArea;

    // Copy the authValue
    nvIndex.authValue = *authValue;

    // Add index to NV memory
    result = NvAdd(entrySize, sizeof(NV_INDEX), TPM_RH_UNASSIGNED, (BYTE*)&nvIndex);
    if(result == TPM_RC_SUCCESS)
    {
        // If the data of NV Index is RAM backed, add the data area in RAM as well
        if(IS_ATTRIBUTE(publicArea->attributes, TPMA_NV, ORDERLY))
            NvAddRAM(publicArea);
    }
    return result;
}

//*** NvAddEvictObject()
// This function is used to assign NV memory to a persistent object.
//  Return Type: TPM_RC
//      TPM_RC_NV_HANDLE        the requested handle is already in use
//      TPM_RC_NV_SPACE         insufficient NV space
TPM_RC
NvAddEvictObject(TPMI_DH_OBJECT evictHandle,  // IN: new evict handle
                 OBJECT*        object        // IN: object to be added
)
{
    TPM_HANDLE temp = object->evictHandle;
```

```
    TPM_RC     result;
    //
    // Check if we have enough space to add the evict object
    // An evict object needs 8 bytes in index table + sizeof OBJECT
    // In this implementation, the only resource limitation is the available NV
    // space.  Other implementation may have other limitation on evict object
    // handle space
    if(!NvTestSpace(sizeof(OBJECT) + sizeof(TPM_HANDLE), FALSE, FALSE))
        return TPM_RC_NV_SPACE;

    // Set evict attribute and handle
    object->attributes.evict = SET;
    object->evictHandle      = evictHandle;

    // Now put this in NV
    result = NvAdd(sizeof(OBJECT), sizeof(OBJECT), evictHandle, (BYTE*)object);

    // Put things back the way they were
    object->attributes.evict = CLEAR;
    object->evictHandle      = temp;

    return result;
}

//*** NvDeleteIndex()
// This function is used to delete an NV Index.
//  Return Type: TPM_RC
//      TPM_RC_NV_UNAVAILABLE   NV is not accessible
//      TPM_RC_NV_RATE          NV is rate limiting
TPM_RC
NvDeleteIndex(NV_INDEX* nvIndex,    // IN: an in RAM index descriptor
              NV_REF    entityAddr  // IN: location in NV
)
{
    TPM_RC result;
    //
    if(nvIndex != NULL)
    {
        // Whenever a counter is deleted, make sure that the MaxCounter value is
        // updated to reflect the value
        if(IsNvCounterIndex(nvIndex->publicArea.attributes)
           && IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
            NvUpdateMaxCount(NvGetUINT64Data(nvIndex, entityAddr));
        result = NvDelete(entityAddr);
        if(result != TPM_RC_SUCCESS)
            return result;
        // If the NV Index is RAM backed, delete the RAM data as well
        if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, ORDERLY))
            NvDeleteRAM(nvIndex->publicArea.nvIndex);
        NvIndexCacheInit();
    }
    return TPM_RC_SUCCESS;
}

//*** NvDeleteEvict()
// This function will delete a NV evict object.
// Will return success if object deleted or if it does not exist

TPM_RC
NvDeleteEvict(TPM_HANDLE handle  // IN: handle of entity to be deleted
)
{
    NV_REF entityAddr = NvFindEvict(handle, NULL);  // pointer to entity
    TPM_RC result     = TPM_RC_SUCCESS;
    //
    if(entityAddr != 0)
```

```
        result = NvDelete(entityAddr);
    return result;
}

//*** NvFlushHierarchy()
// This function will delete persistent objects belonging to the indicated hierarchy.
// If the storage hierarchy is selected, the function will also delete any
// NV Index defined using ownerAuth.
//  Return Type: TPM_RC
//      TPM_RC_NV_RATE          NV is unavailable because of rate limit
//      TPM_RC_NV_UNAVAILABLE   NV is inaccessible
TPM_RC
NvFlushHierarchy(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy to be flushed.
)
{
    NV_REF     iter = NV_REF_INIT;
    NV_REF     currentAddr;
    TPM_HANDLE entityHandle;
    TPM_RC     result = TPM_RC_SUCCESS;
    //
    while((currentAddr = NvNext(&iter, &entityHandle)) != 0)
    {
        if(HandleGetType(entityHandle) == TPM_HT_NV_INDEX)
        {
            NV_INDEX nvIndex;
            //
            // If flush endorsement or platform hierarchy, no NV Index would be
            // flushed
            if(hierarchy == TPM_RH_ENDORSEMENT || hierarchy == TPM_RH_PLATFORM)
                continue;
            // Get the index information
            NvReadNvIndexInfo(currentAddr, &nvIndex);

            // For storage hierarchy, flush OwnerCreated index
            if(!IS_ATTRIBUTE(nvIndex.publicArea.attributes, TPMA_NV, PLATFORMCREATE))
            {
                // Delete the index (including RAM for orderly)
                result = NvDeleteIndex(&nvIndex, currentAddr);
                if(result != TPM_RC_SUCCESS)
                    break;
                // Re-iterate from beginning after a delete
                iter = NV_REF_INIT;
            }
        }
        else if(HandleGetType(entityHandle) == TPM_HT_PERSISTENT)
        {
            OBJECT_ATTRIBUTES attributes;
            //
            NvRead(&attributes,
                   (UINT32)(currentAddr + sizeof(TPM_HANDLE)
                            + offsetof(OBJECT, attributes)),
                   sizeof(OBJECT_ATTRIBUTES));
            // If the evict object belongs to the hierarchy to be flushed...
            if((hierarchy == TPM_RH_PLATFORM && attributes.ppsHierarchy == SET)
               || (hierarchy == TPM_RH_OWNER && attributes.spsHierarchy == SET)
               || (hierarchy == TPM_RH_ENDORSEMENT && attributes.epsHierarchy == SET))
            {
                // ...then delete the evict object
                result = NvDelete(currentAddr);
                if(result != TPM_RC_SUCCESS)
                    break;
                // Re-iterate from beginning after a delete
                iter = NV_REF_INIT;
            }
        }
        else
```

```
        {
            FAIL(FATAL_ERROR_INTERNAL);
        }
    }
    return result;
}

//*** NvSetGlobalLock()
// This function is used to SET the TPMA_NV_WRITELOCKED attribute for all
// NV indexes that have TPMA_NV_GLOBALLOCK SET. This function is use by
// TPM2_NV_GlobalWriteLock().
//  Return Type: TPM_RC
//      TPM_RC_NV_RATE          NV is unavailable because of rate limit
//      TPM_RC_NV_UNAVAILABLE   NV is inaccessible
TPM_RC
NvSetGlobalLock(void)
{
    NV_REF     iter    = NV_REF_INIT;
    NV_RAM_REF ramIter = NV_RAM_REF_INIT;
    NV_REF     currentAddr;
    NV_RAM_REF currentRamAddr;
    TPM_RC     result = TPM_RC_SUCCESS;
    //
    // Check all normal indexes
    while((currentAddr = NvNextIndex(NULL, &iter)) != 0)
    {
        TPMA_NV attributes = NvReadNvIndexAttributes(currentAddr);
        //
        // See if it should be locked
        if(!IS_ATTRIBUTE(attributes, TPMA_NV, ORDERLY)
           && IS_ATTRIBUTE(attributes, TPMA_NV, GLOBALLOCK))
        {
            SET_ATTRIBUTE(attributes, TPMA_NV, WRITELOCKED);
            result = NvWriteNvIndexAttributes(currentAddr, attributes);
            if(result != TPM_RC_SUCCESS)
                return result;
        }
    }
    // Now search all the orderly attributes
    while((currentRamAddr = NvRamNext(&ramIter, NULL)) != 0)
    {
        // See if it should be locked
        TPMA_NV attributes = NvReadRamIndexAttributes(currentRamAddr);
        if(IS_ATTRIBUTE(attributes, TPMA_NV, GLOBALLOCK))
        {
            SET_ATTRIBUTE(attributes, TPMA_NV, WRITELOCKED);
            NvWriteRamIndexAttributes(currentRamAddr, attributes);
        }
    }
    return result;
}

//***InsertSort()
// Sort a handle into handle list in ascending order.  The total handle number in
// the list should not exceed MAX_CAP_HANDLES
static void InsertSort(TPML_HANDLE* handleList,  // IN/OUT: sorted handle list
                       UINT32       count,  // IN: maximum count in the handle list
                       TPM_HANDLE   entityHandle  // IN: handle to be inserted
)
{
    UINT32 i, j;
    UINT32 originalCount;
    //
    // For a corner case that the maximum count is 0, do nothing
    if(count == 0)
        return;
```

```
    // For empty list, add the handle at the beginning and return
    if(handleList->count == 0)
    {
        handleList->handle[0] = entityHandle;
        handleList->count++;
        return;
    }
    // Check if the maximum of the list has been reached
    originalCount = handleList->count;
    if(originalCount < count)
        handleList->count++;
    // Insert the handle to the list
    for(i = 0; i < originalCount; i++)
    {
        if(handleList->handle[i] > entityHandle)
        {
            for(j = handleList->count - 1; j > i; j--)
            {
                handleList->handle[j] = handleList->handle[j - 1];
            }
            break;
        }
    }
    // If a slot was found, insert the handle in this position
    if(i < originalCount || handleList->count > originalCount)
        handleList->handle[i] = entityHandle;
    return;
}

//*** NvCapGetPersistent()
// This function is used to get a list of handles of the persistent objects,
// starting at 'handle'.
//
// 'Handle' must be in valid persistent object handle range, but does not
// have to reference an existing persistent object.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
NvCapGetPersistent(TPMI_DH_OBJECT handle,  // IN: start handle
                   UINT32         count,   // IN: maximum number of returned handles
                   TPML_HANDLE*   handleList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    NV_REF      iter = NV_REF_INIT;
    NV_REF      currentAddr;
    TPM_HANDLE  entityHandle;
    //
    pAssert(HandleGetType(handle) == TPM_HT_PERSISTENT);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES)
        count = MAX_CAP_HANDLES;

    while((currentAddr = NvNextEvict(&entityHandle, &iter)) != 0)
    {
        // Ignore persistent handles that have values less than the input handle
        if(entityHandle < handle)
            continue;
        // if the handles in the list have reached the requested count, and there
        // are still handles need to be inserted, indicate that there are more.
        if(handleList->count == count)
```

```
            more = YES;
        // A handle with a value larger than start handle is a candidate
        // for return. Insert sort it to the return list.  Insert sort algorithm
        // is chosen here for simplicity based on the assumption that the total
        // number of NV indexes is small.  For an implementation that may allow
        // large number of NV indexes, a more efficient sorting algorithm may be
        // used here.
        InsertSort(handleList, count, entityHandle);
    }
    return more;
}

//*** NvCapGetOnePersistent()
// This function returns whether a given persistent handle exists.
//
// 'Handle' must be in valid persistent object handle range.
BOOL NvCapGetOnePersistent(TPMI_DH_OBJECT handle)  // IN: handle
{
    NV_REF     iter = NV_REF_INIT;
    NV_REF     currentAddr;
    TPM_HANDLE entityHandle;

    pAssert(HandleGetType(handle) == TPM_HT_PERSISTENT);

    while((currentAddr = NvNextEvict(&entityHandle, &iter)) != 0)
    {
        if(entityHandle == handle)
        {
            return TRUE;
        }
    }
    return FALSE;
}

//*** NvCapGetIndex()
// This function returns a list of handles of NV indexes, starting from 'handle'.
// 'Handle' must be in the range of NV indexes, but does not have to reference
// an existing NV Index.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles to report
//      NO          all the available handles has been reported
TPMI_YES_NO
NvCapGetIndex(TPMI_DH_OBJECT handle,     // IN: start handle
              UINT32         count,      // IN: max number of returned handles
              TPML_HANDLE*   handleList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    NV_REF      iter = NV_REF_INIT;
    NV_REF      currentAddr;
    TPM_HANDLE  nvHandle;
    //
    pAssert(HandleGetType(handle) == TPM_HT_NV_INDEX);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES)
        count = MAX_CAP_HANDLES;

    while((currentAddr = NvNextIndex(&nvHandle, &iter)) != 0)
    {
        // Ignore index handles that have values less than the 'handle'
        if(nvHandle < handle)
            continue;
```

```
        // if the count of handles in the list has reached the requested count,
        // and there are still handles to report, set more.
        if(handleList->count == count)
            more = YES;
        // A handle with a value larger than start handle is a candidate
        // for return. Insert sort it to the return list.  Insert sort algorithm
        // is chosen here for simplicity based on the assumption that the total
        // number of NV indexes is small.  For an implementation that may allow
        // large number of NV indexes, a more efficient sorting algorithm may be
        // used here.
        InsertSort(handleList, count, nvHandle);
    }
    return more;
}

//*** NvCapGetOneIndex()
// This function whether an NV index exists.
BOOL NvCapGetOneIndex(TPMI_DH_OBJECT handle)  // IN: handle
{
    NV_REF     iter = NV_REF_INIT;
    NV_REF     currentAddr;
    TPM_HANDLE nvHandle;

    pAssert(HandleGetType(handle) == TPM_HT_NV_INDEX);

    while((currentAddr = NvNextIndex(&nvHandle, &iter)) != 0)
    {
        if(nvHandle == handle)
        {
            return TRUE;
        }
    }
    return FALSE;
}

//*** NvCapGetIndexNumber()
// This function returns the count of NV Indexes currently defined.
UINT32
NvCapGetIndexNumber(void)
{
    UINT32 num  = 0;
    NV_REF iter = NV_REF_INIT;
    //
    while(NvNextIndex(NULL, &iter) != 0)
        num++;
    return num;
}

//*** NvCapGetPersistentNumber()
// Function returns the count of persistent objects currently in NV memory.
UINT32
NvCapGetPersistentNumber(void)
{
    UINT32     num  = 0;
    NV_REF     iter = NV_REF_INIT;
    TPM_HANDLE handle;
    //
    while(NvNextEvict(&handle, &iter) != 0)
        num++;
    return num;
}

//*** NvCapGetPersistentAvail()
// This function returns an estimate of the number of additional persistent
// objects that could be loaded into NV memory.
UINT32
```

```
NvCapGetPersistentAvail(void)
{
    UINT32 availNVSpace;
    UINT32 counterNum = NvCapGetCounterNumber();
    UINT32 reserved   = sizeof(NV_LIST_TERMINATOR);
    //
    // Get the available space in NV storage
    availNVSpace = NvGetFreeBytes();

    if(counterNum < MIN_COUNTER_INDICES)
    {
        // Some space has to be reserved for counter objects.
        reserved += (MIN_COUNTER_INDICES - counterNum) * NV_INDEX_COUNTER_SIZE;
        if(reserved > availNVSpace)
            availNVSpace = 0;
        else
            availNVSpace -= reserved;
    }
    return availNVSpace / NV_EVICT_OBJECT_SIZE;
}

//*** NvCapGetCounterNumber()
// Get the number of defined NV Indexes that are counter indexes.
UINT32
NvCapGetCounterNumber(void)
{
    NV_REF iter = NV_REF_INIT;
    NV_REF currentAddr;
    UINT32 num = 0;
    //
    while((currentAddr = NvNextIndex(NULL, &iter)) != 0)
    {
        TPMA_NV attributes = NvReadNvIndexAttributes(currentAddr);
        if(IsNvCounterIndex(attributes))
            num++;
    }
    return num;
}

//*** NvSetStartupAttributes()
// Local function to set the attributes of an Index at TPM Reset and TPM Restart.
static TPMA_NV NvSetStartupAttributes(TPMA_NV attributes,  // IN: attributes to change
                                      STARTUP_TYPE type    // IN: start up type
)
{
    // Clear read lock
    CLEAR_ATTRIBUTE(attributes, TPMA_NV, READLOCKED);

    // Will change a non counter index to the unwritten state if:
    // a) TPMA_NV_CLEAR_STCLEAR is SET
    // b) orderly and TPM Reset
    if(!IsNvCounterIndex(attributes))
    {
        if(IS_ATTRIBUTE(attributes, TPMA_NV, CLEAR_STCLEAR)
           || (IS_ATTRIBUTE(attributes, TPMA_NV, ORDERLY) && (type == SU_RESET)))
            CLEAR_ATTRIBUTE(attributes, TPMA_NV, WRITTEN);
    }
    // Unlock any index that is not written or that does not have
    // TPMA_NV_WRITEDEFINE SET.
    if(!IS_ATTRIBUTE(attributes, TPMA_NV, WRITTEN)
       || !IS_ATTRIBUTE(attributes, TPMA_NV, WRITEDEFINE))
        CLEAR_ATTRIBUTE(attributes, TPMA_NV, WRITELOCKED);
    return attributes;
}

//*** NvEntityStartup()
```

```
//  This function is called at TPM_Startup(). If the startup completes
//  a TPM Resume cycle, no action is taken. If the startup is a TPM Reset
//  or a TPM Restart, then this function will:
//  a) clear read/write lock;
//  b) reset NV Index data that has TPMA_NV_CLEAR_STCLEAR SET; and
//  c) set the lower bits in orderly counters to 1 for a non-orderly startup
//
//  It is a prerequisite that NV be available for writing before this
//  function is called.
BOOL NvEntityStartup(STARTUP_TYPE type  // IN: start up type
)
{
    NV_REF     iter    = NV_REF_INIT;
    NV_RAM_REF ramIter = NV_RAM_REF_INIT;
    NV_REF     currentAddr;  // offset points to the current entity
    NV_RAM_REF currentRamAddr;
    TPM_HANDLE nvHandle;
    TPMA_NV    attributes;
    //
    // Restore RAM index data
    NvRead(s_indexOrderlyRam, NV_INDEX_RAM_DATA, sizeof(s_indexOrderlyRam));

    // Initialize the max NV counter value
    NvSetMaxCount(NvGetMaxCount());

    // If recovering from state save, do nothing else
    if(type == SU_RESUME)
        return TRUE;
    // Iterate all the NV Index to clear the locks
    while((currentAddr = NvNextIndex(&nvHandle, &iter)) != 0)
    {
        attributes = NvReadNvIndexAttributes(currentAddr);

        // If this is an orderly index, defer processing until loop below
        if(IS_ATTRIBUTE(attributes, TPMA_NV, ORDERLY))
            continue;
        // Set the attributes appropriate for this startup type
        attributes = NvSetStartupAttributes(attributes, type);
        NvWriteNvIndexAttributes(currentAddr, attributes);
    }
    // Iterate all the orderly indexes to clear the locks and initialize counters
    while((currentRamAddr = NvRamNext(&ramIter, NULL)) != 0)
    {
        attributes = NvReadRamIndexAttributes(currentRamAddr);

        attributes = NvSetStartupAttributes(attributes, type);

        // update attributes in RAM
        NvWriteRamIndexAttributes(currentRamAddr, attributes);

        // Set the lower bits in an orderly counter to 1 for a non-orderly startup
        if(IsNvCounterIndex(attributes) && (g_prevOrderlyState == SU_NONE_VALUE))
        {
            UINT64 counter;
            //
            // Read the counter value last saved to NV.
            counter = BYTE_ARRAY_TO_UINT64(currentRamAddr + sizeof(NV_RAM_HEADER));

            // Set the lower bits of counter to 1's
            counter |= MAX_ORDERLY_COUNT;

            // Write back to RAM
            // NOTE: Do not want to force a write to NV here. The counter value will
            // stay in RAM until the next shutdown or rollover.
            UINT64_TO_BYTE_ARRAY(counter, currentRamAddr + sizeof(NV_RAM_HEADER));
        }
```

```
    }
    return TRUE;
}

//*** NvCapGetCounterAvail()
// This function returns an estimate of the number of additional counter type
// NV indexes that can be defined.
UINT32
NvCapGetCounterAvail(void)
{
    UINT32 availNVSpace;
    UINT32 availRAMSpace;
    UINT32 persistentNum = NvCapGetPersistentNumber();
    UINT32 reserved      = sizeof(NV_LIST_TERMINATOR);
    //
    // Get the available space in NV storage
    availNVSpace = NvGetFreeBytes();

    if(persistentNum < MIN_EVICT_OBJECTS)
    {
        // Some space has to be reserved for evict object. Adjust availNVSpace.
        reserved += (MIN_EVICT_OBJECTS - persistentNum) * NV_EVICT_OBJECT_SIZE;
        if(reserved > availNVSpace)
            availNVSpace = 0;
        else
            availNVSpace -= reserved;
    }
    // Compute the available space in RAM
    availRAMSpace = (int)(RAM_ORDERLY_END - NvRamGetEnd());

    // Return the min of counter number in NV and in RAM
    if(availNVSpace / NV_INDEX_COUNTER_SIZE
       > availRAMSpace / NV_RAM_INDEX_COUNTER_SIZE)
        return availRAMSpace / NV_RAM_INDEX_COUNTER_SIZE;
    else
        return availNVSpace / NV_INDEX_COUNTER_SIZE;
}

//*** NvFindHandle()
// this function returns the offset in NV memory of the entity associated
// with the input handle.  A value of zero indicates that handle does not
//  exist reference an existing persistent object or defined NV Index.
NV_REF
NvFindHandle(TPM_HANDLE handle)
{
    NV_REF     addr;
    NV_REF     iter = NV_REF_INIT;
    TPM_HANDLE nextHandle;
    //
    while((addr = NvNext(&iter, &nextHandle)) != 0)
    {
        if(nextHandle == handle)
            break;
    }
    return addr;
}

//** NV Max Counter
//*** Introduction
// The TPM keeps track of the highest value of a deleted counter index. When an
// index is deleted, this value is updated if the deleted counter index is greater
// than the previous value. When a new index is created and first incremented, it
// will get a value that is at least one greater than any other index than any
// previously deleted index. This insures that it is not possible to roll back an
// index.
//
```

```
// The highest counter value is kept in NV in a special end-of-list marker. This
// marker is only updated when an index is deleted. Otherwise it just moves.
//
// When the TPM starts up, it searches NV for the end of list marker and initializes
// an in memory value (s_maxCounter).

//*** NvReadMaxCount()
// This function returns the max NV counter value.
//
UINT64
NvReadMaxCount(void)
{
    return s_maxCounter;
}

//*** NvUpdateMaxCount()
// This function updates the max counter value to NV memory. This is just staging
// for the actual write that will occur when the NV index memory is modified.
//
void NvUpdateMaxCount(UINT64 count)
{
    if(count > s_maxCounter)
        s_maxCounter = count;
}

//*** NvSetMaxCount()
// This function is used at NV initialization time to set the initial value of
// the maximum counter.
void NvSetMaxCount(UINT64 value)
{
    s_maxCounter = value;
}

//*** NvGetMaxCount()
// Function to get the NV max counter value from the end-of-list marker
UINT64
NvGetMaxCount(void)
{
    NV_REF iter = NV_REF_INIT;
    NV_REF currentAddr;
    UINT64 maxCount;
    //
    // Find the end of list marker and initialize the NV Max Counter value.
    while((currentAddr = NvNext(&iter, NULL)) != 0)
        ;
    // 'iter' should be pointing at the end of list marker so read in the current
    // value of the s_maxCounter.
    NvRead(&maxCount, iter + sizeof(UINT32), sizeof(maxCount));

    return maxCount;
}
```

7.174 /tpm/src/subsystem/NvReserved.c

```
//** Introduction

// The NV memory is divided into two areas: dynamic space for user defined NV
// Indices and evict objects, and reserved space for TPM persistent and state save
// data.
//
// The entries in dynamic space are a linked list of entries. Each entry has, as its
// first field, a size. If the size field is zero, it marks the end of the
// list.
//
// An allocation of an Index or evict object may use almost all of the remaining
```

```
// NV space such that the size field will not fit. The functions that search the
// list are aware of this and will terminate the search if they either find a zero
// size or recognize that there is insufficient space for the size field.
//
// An Index allocation will contain an NV_INDEX structure. If the Index does not
// have the orderly attribute, the NV_INDEX is followed immediately by the NV data.
//
// An evict object entry contains a handle followed by an OBJECT structure. This
// results in both the Index and Evict Object having an identifying handle as the
// first field following the size field.
//
// When an Index has the orderly attribute, the data is kept in RAM. This RAM is
// saved to backing store in NV memory on any orderly shutdown. The entries in
// orderly memory are also a linked list using a size field as the first entry. As
// with the NV memory, the list is terminated by a zero size field or when the last
// entry leaves insufficient space for the terminating size field.
//
// The attributes of an orderly index are maintained in RAM memory in order to
// reduce the number of NV writes needed for orderly data. When an orderly index
// is created, an entry is made in the dynamic NV memory space that holds the Index
// authorizations (authPolicy and authValue) and the size of the data. This entry is
// only modified if the authValue  of the index is changed. The more volatile data
// of the index is kept in RAM. When an orderly Index is created or deleted, the
// RAM data is copied to NV backing store so that the image in the backing store
// matches the layout of RAM. In normal operation. The RAM data is also copied on
// any orderly shutdown. In normal operation, the only other reason for writing
// to the backing store for RAM is when a counter is first written (TPMA_NV_WRITTEN
// changes from CLEAR to SET) or when a counter "rolls over."
//
// Static space contains items that are individually modifiable. The values are in
// the 'gp' PERSISTENT_DATA structure in RAM and mapped to locations in NV.
//

//** Includes, Defines
#define NV_C
#include "Tpm.h"

//**********************************************
//** Functions
//**********************************************

//*** NvInitStatic()
// This function initializes the static variables used in the NV subsystem.
static void NvInitStatic(void)
{
    // In some implementations, the end of NV is variable and is set at boot time.
    // This value will be the same for each boot, but is not necessarily known
    // at compile time.
    s_evictNvEnd = (NV_REF)NV_MEMORY_SIZE;
    return;
}

//*** NvCheckState()
// Function to check the NV state by accessing the platform-specific function
// to get the NV state.  The result state is registered in s_NvIsAvailable
// that will be reported by NvIsAvailable.
//
// This function is called at the beginning of ExecuteCommand before any potential
// check of g_NvStatus.
void NvCheckState(void)
{
    int func_return;
    //
    func_return = _plat__GetNvReadyState();
    if(func_return == NV_READY)
    {
```

```
        g_NvStatus = TPM_RC_SUCCESS;
    }
    else if(func_return == NV_WRITEFAILURE)
    {
        g_NvStatus = TPM_RC_NV_UNAVAILABLE;
    }
    else
    {
        // if(func_return == NV_RATE_LIMIT) or anything else
        // assume retry later might work
        g_NvStatus = TPM_RC_NV_RATE;
    }

    return;
}

//*** NvCommit
// This is a wrapper for the platform function to commit pending NV writes.
BOOL NvCommit(void)
{
    return (_plat__NvCommit() == 0);
}

//*** NvPowerOn()
//  This function is called at _TPM_Init to initialize the NV environment.
//  Return Type: BOOL
//      TRUE(1)         all NV was initialized
//      FALSE(0)        the NV containing saved state had an error and
//                      TPM2_Startup(CLEAR) is required
BOOL NvPowerOn(void)
{
    int nvError = 0;
    // If power was lost, need to re-establish the RAM data that is loaded from
    // NV and initialize the static variables
    if(g_powerWasLost)
    {
        if((nvError = _plat__NVEnable(NULL, 0)) < 0)
            FAIL(FATAL_ERROR_NV_UNRECOVERABLE);
        NvInitStatic();
    }
    return nvError == 0;
}

//*** NvManufacture()
// This function initializes the NV system at pre-install time.
//
// This function should only be called in a manufacturing environment or in a
// simulation.
//
// The layout of NV memory space is an implementation choice.
void NvManufacture(void)
{
#if SIMULATION
    // Simulate the NV memory being in the erased state.
    _plat__NvMemoryClear(0, NV_MEMORY_SIZE);
#endif
    // Initialize static variables
    NvInitStatic();
    // Clear the RAM used for Orderly Index data
    MemorySet(s_indexOrderlyRam, 0, RAM_INDEX_SPACE);
    // Write that Orderly Index data to NV
    NvUpdateIndexOrderlyData();
    // Initialize the next offset of the first entry in evict/index list to 0 (the
    // end of list marker) and the initial s_maxCounterValue;
    NvSetMaxCount(0);
    // Put the end of list marker at the end of memory. This contains the MaxCount
```

```
    // value as well as the end marker.
    NvWriteNvListEnd(NV_USER_DYNAMIC);
    return;
}

//*** NvRead()
// This function is used to move reserved data from NV memory to RAM.
void NvRead(void*  outBuffer,  // OUT: buffer to receive data
            UINT32 nvOffset,   // IN: offset in NV of value
            UINT32 size        // IN: size of the value to read
)
{
    // Input type should be valid
    pAssert(nvOffset + size < NV_MEMORY_SIZE);
    _plat__NvMemoryRead(nvOffset, size, outBuffer);
    return;
}

//*** NvWrite()
// This function is used to post reserved data for writing to NV memory. Before
// the TPM completes the operation, the value will be written.
BOOL NvWrite(UINT32 nvOffset,  // IN: location in NV to receive data
             UINT32 size,      // IN: size of the data to move
             void*  inBuffer   // IN: location containing data to write
)
{
    // Input type should be valid
    if(nvOffset + size <= NV_MEMORY_SIZE)
    {
        // Set the flag that a NV write happened
        SET_NV_UPDATE(UT_NV);
        return _plat__NvMemoryWrite(nvOffset, size, inBuffer);
    }
    return FALSE;
}

//*** NvUpdatePersistent()
// This function is used to update a value in the PERSISTENT_DATA structure and
// commits the value to NV.
void NvUpdatePersistent(
    UINT32 offset,  // IN: location in PERMANENT_DATA to be updated
    UINT32 size,    // IN: size of the value
    void*  buffer   // IN: the new data
)
{
    pAssert(offset + size <= sizeof(gp));
    MemoryCopy(&gp + offset, buffer, size);
    NvWrite(offset, size, buffer);
}

//*** NvClearPersistent()
// This function is used to clear a persistent data entry and commit it to NV
void NvClearPersistent(UINT32 offset,  // IN: the offset in the PERMANENT_DATA
                                       //     structure to be cleared (zeroed)
                       UINT32 size     // IN: number of bytes to clear
)
{
    pAssert(offset + size <= sizeof(gp));
    MemorySet((&gp) + offset, 0, size);
    NvWrite(offset, size, (&gp) + offset);
}

//*** NvReadPersistent()
// This function reads persistent data to the RAM copy of the 'gp' structure.
void NvReadPersistent(void)
{
```

```
    NvRead(&gp, NV_PERSISTENT_DATA, sizeof(gp));
    return;
}
```

7.175 /tpm/src/subsystem/Object.c

```
//** Introduction
// This file contains the functions that manage the object store of the TPM.

//** Includes and Data Definitions
#define OBJECT_C

#include "Tpm.h"
#include "Marshal.h"

//** Functions

//*** ObjectFlush()
// This function marks an object slot as available.
// Since there is no checking of the input parameters, it should be used
// judiciously.
// Note: This could be converted to a macro.
void ObjectFlush(OBJECT* object)
{
    object->attributes.occupied = CLEAR;
}

//*** ObjectSetInUse()
// This access function sets the occupied attribute of an object slot.
void ObjectSetInUse(OBJECT* object)
{
    object->attributes.occupied = SET;
}

//*** ObjectStartup()
// This function is called at TPM2_Startup() to initialize the object subsystem.
BOOL ObjectStartup(void)
{
    UINT32 i;
    //
    // object slots initialization
    for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    {
        //Set the slot to not occupied
        ObjectFlush(&s_objects[i]);
    }
    return TRUE;
}

//*** ObjectCleanupEvict()
//
// In this implementation, a persistent object is moved from NV into an object slot
// for processing. It is flushed after command execution. This function is called
// from ExecuteCommand().
void ObjectCleanupEvict(void)
{
    UINT32 i;
    //
    // This has to be iterated because a command may have two handles
    // and they may both be persistent.
    // This could be made to be more efficient so that a search is not needed.
    for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    {
        // If an object is a temporary evict object, flush it from slot
        OBJECT* object = &s_objects[i];
```

```
        if(object->attributes.evict == SET)
            ObjectFlush(object);
    }
    return;
}

//*** IsObjectPresent()
// This function checks to see if a transient handle references a loaded
// object.  This routine should not be called if the handle is not a
// transient handle. The function validates that the handle is in the
// implementation-dependent allowed in range for loaded transient objects.
//  Return Type: BOOL
//      TRUE(1)         handle references a loaded object
//      FALSE(0)        handle is not an object handle, or it does not
//                      reference to a loaded object
BOOL IsObjectPresent(TPMI_DH_OBJECT handle  // IN: handle to be checked
)
{
    UINT32 slotIndex = handle - TRANSIENT_FIRST;
    // Since the handle is just an index into the array that is zero based, any
    // handle value outsize of the range of:
    //    TRANSIENT_FIRST -- (TRANSIENT_FIRST + MAX_LOADED_OBJECT - 1)
    // will now be greater than or equal to MAX_LOADED_OBJECTS
    if(slotIndex >= MAX_LOADED_OBJECTS)
        return FALSE;
    // Indicate if the slot is occupied
    return (s_objects[slotIndex].attributes.occupied == TRUE);
}

//*** ObjectIsSequence()
// This function is used to check if the object is a sequence object. This function
// should not be called if the handle does not reference a loaded object.
//  Return Type: BOOL
//      TRUE(1)         object is an HMAC, hash, or event sequence object
//      FALSE(0)        object is not an HMAC, hash, or event sequence object
BOOL ObjectIsSequence(OBJECT* object  // IN: handle to be checked
)
{
    pAssert(object != NULL);
    return (object->attributes.hmacSeq == SET || object->attributes.hashSeq == SET
            || object->attributes.eventSeq == SET);
}

//*** HandleToObject()
// This function is used to find the object structure associated with a handle.
//
// This function requires that 'handle' references a loaded object or a permanent
// handle.
OBJECT* HandleToObject(TPMI_DH_OBJECT handle  // IN: handle of the object
)
{
    UINT32 index;
    //
    // Return NULL if the handle references a permanent handle because there is no
    // associated OBJECT.
    if(HandleGetType(handle) == TPM_HT_PERMANENT)
        return NULL;
    // In this implementation, the handle is determined by the slot occupied by the
    // object.
    index = handle - TRANSIENT_FIRST;
    pAssert(index < MAX_LOADED_OBJECTS);
    pAssert(s_objects[index].attributes.occupied);
    return &s_objects[index];
}

//*** GetQualifiedName()
```

```
// This function returns the Qualified Name of the object. In this implementation,
// the Qualified Name is computed when the object is loaded and is saved in the
// internal representation of the object. The alternative would be to retain the
// Name of the parent and compute the QN when needed. This would take the same
// amount of space so it is not recommended that the alternate be used.
//
// This function requires that 'handle' references a loaded object.
void GetQualifiedName(TPMI_DH_OBJECT handle,     // IN: handle of the object
                      TPM2B_NAME* qualifiedName  // OUT: qualified name of the object
)
{
    OBJECT* object;
    //
    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
            qualifiedName->t.size = sizeof(TPM_HANDLE);
            UINT32_TO_BYTE_ARRAY(handle, qualifiedName->t.name);
            break;
        case TPM_HT_TRANSIENT:
            object = HandleToObject(handle);
            if(object == NULL || object->publicArea.nameAlg == TPM_ALG_NULL)
                qualifiedName->t.size = 0;
            else
                // Copy the name
                *qualifiedName = object->qualifiedName;
            break;
        default:
            FAIL(FATAL_ERROR_INTERNAL);
    }
    return;
}

//*** GetHierarchy()
// This function returns the handle of the hierarchy to which a handle belongs.
//
// This function requires that 'handle' references a loaded object.
TPMI_RH_HIERARCHY
GetHierarchy(TPMI_DH_OBJECT handle  // IN :object handle
)
{
    return HandleToObject(handle)->hierarchy;
}

//*** FindEmptyObjectSlot()
// This function finds an open object slot, if any. It will clear the attributes
// but will not set the occupied attribute. This is so that a slot may be used
// and discarded if everything does not go as planned.
//  Return Type: OBJECT *
//      NULL        no open slot found
//      != NULL     pointer to available slot
OBJECT* FindEmptyObjectSlot(TPMI_DH_OBJECT* handle  // OUT: (optional)
)
{
    UINT32  i;
    OBJECT* object;
    //
    for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    {
        object = &s_objects[i];
        if(object->attributes.occupied == CLEAR)
        {
            if(handle)
                *handle = i + TRANSIENT_FIRST;
            // Initialize the object attributes
            MemorySet(&object->attributes, 0, sizeof(OBJECT_ATTRIBUTES));
```

```
            object->hierarchy = TPM_RH_NULL;
            return object;
        }
    }
    return NULL;
}

//*** ObjectAllocateSlot()
// This function is used to allocate a slot in internal object array.
OBJECT* ObjectAllocateSlot(TPMI_DH_OBJECT* handle  // OUT: handle of allocated object
)
{
    OBJECT* object = FindEmptyObjectSlot(handle);
    //
    if(object != NULL)
    {
        // if found, mark as occupied
        ObjectSetInUse(object);
    }
    return object;
}

//*** ObjectSetLoadedAttributes()
// This function sets the internal attributes for a loaded object. It is called to
// finalize the OBJECT attributes (not the TPMA_OBJECT attributes) for a loaded
// object.
void ObjectSetLoadedAttributes(OBJECT* object,  // IN: object attributes to finalize
                               TPM_HANDLE parentHandle  // IN: the parent handle
)
{
    OBJECT*     parent           = HandleToObject(parentHandle);
    TPMA_OBJECT objectAttributes = object->publicArea.objectAttributes;
    //
    // Copy the stClear attribute from the public area. This could be overwritten
    // if the parent has stClear SET
    object->attributes.stClear = IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, stClear);
    // If parent handle is a permanent handle, it is a primary (unless it is NULL
    if(parent == NULL)
    {
        object->hierarchy          = parentHandle;
        object->attributes.primary = SET;
        switch(HierarchyNormalizeHandle(object->hierarchy))
        {
            case TPM_RH_ENDORSEMENT:
                object->attributes.epsHierarchy = SET;
                break;
            case TPM_RH_OWNER:
                object->attributes.spsHierarchy = SET;
                break;
            case TPM_RH_PLATFORM:
                object->attributes.ppsHierarchy = SET;
                break;
            default:
                // Treat the temporary attribute as a hierarchy
                object->attributes.temporary = SET;
                object->attributes.primary   = CLEAR;
                break;
        }
    }
    else
    {
        // is this a stClear object
        object->attributes.stClear =
            (IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, stClear)
             || (parent->attributes.stClear == SET));
        object->attributes.epsHierarchy = parent->attributes.epsHierarchy;
```

```
        object->attributes.spsHierarchy = parent->attributes.spsHierarchy;
        object->attributes.ppsHierarchy = parent->attributes.ppsHierarchy;
        // An object is temporary if its parent is temporary or if the object
        // is external
        object->attributes.temporary = parent->attributes.temporary
                                     || object->attributes.external;
        object->hierarchy = parent->hierarchy;
    }
    // If this is an external object, set the QN == name but don't SET other
    // key properties ('parent' or 'derived')
    if(object->attributes.external)
        object->qualifiedName = object->name;
    else
    {
        // check attributes for different types of parents
        if(IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, restricted)
           && !object->attributes.publicOnly
           && IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, decrypt)
           && object->publicArea.nameAlg != TPM_ALG_NULL)
        {
            // This is a parent. If it is not a KEYEDHASH, it is an ordinary parent.
            // Otherwise, it is a derivation parent.
            if(object->publicArea.type == TPM_ALG_KEYEDHASH)
                object->attributes.derivation = SET;
            else
                object->attributes.isParent = SET;
        }
        ComputeQualifiedName(parentHandle,
                             object->publicArea.nameAlg,
                             &object->name,
                             &object->qualifiedName);
    }
    // Set slot occupied
    ObjectSetInUse(object);
    return;
}

//*** ObjectLoad()
// Common function to load a non-primary object (i.e., either an Ordinary Object,
// or an External Object). A loaded object has its public area validated
// (unless its 'nameAlg' is TPM_ALG_NULL). If a sensitive part is loaded, it is
// verified to be correct and if both public and sensitive parts are loaded, then
// the cryptographic binding between the objects is validated. This function does
// not cause the allocated slot to be marked as in use.
TPM_RC
ObjectLoad(OBJECT* object,              // IN: pointer to object slot
                                        //     object
           OBJECT*      parent,         // IN: (optional) the parent object
           TPMT_PUBLIC* publicArea,     // IN: public area to be installed in the object
           TPMT_SENSITIVE* sensitive,   // IN: (optional) sensitive area to be
                                        //      installed in the object
           TPM_RC blamePublic,          // IN: parameter number to associate with the
                                        //     publicArea errors
           TPM_RC blameSensitive,       // IN: parameter number to associate with the
                                        //     sensitive area errors
           TPM2B_NAME* name             // IN: (optional)
)
{
    TPM_RC result = TPM_RC_SUCCESS;
    //
    // Do validations of public area object descriptions
    pAssert(publicArea != NULL);

    // Is this public only or a no-name object?
    if(sensitive == NULL || publicArea->nameAlg == TPM_ALG_NULL)
    {
```

```
        // Need to have schemes checked so that we do the right thing with the
        // public key.
        result = SchemeChecks(NULL, publicArea);
    }
    else
    {
        // For any sensitive area, make sure that the seedSize is no larger than the
        // digest size of nameAlg
        if(sensitive->seedValue.t.size > CryptHashGetDigestSize(publicArea->nameAlg))
            return TPM_RCS_KEY_SIZE + blameSensitive;
        // Check attributes and schemes for consistency
        // For the purposes of attributes validation on this non-primary object,
        // either:
        // - parent is not NULL and therefore its attributes are checked for
        //   consistency with the parent, OR
        // - parent is NULL but the object is not a primary object, either
        result =
            PublicAttributesValidation(parent, /*primaryHierarchy = */ 0, publicArea);
    }
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, blamePublic);

    // Sensitive area and binding checks

    // On load, check nothing if the parent is fixedTPM.
    // If the parent is fixedTPM, then this TPM produced this key blob (either
    // by import, or creation). If the parent is not fixedTPM, then an external
    // copy of the parent's protection seed might have been used to create the
    // blob, and we have to validate it.
    // NOTE: By the time a TPMT_SENSITIVE has been decrypted and passed to this
    // function, it has been validated against the corresponding TPMT_PUBLIC.
    // For more information about this check, see PrivateToSensitive.
    if((parent == NULL)
       || ((parent != NULL)
           && !IS_ATTRIBUTE(
               parent->publicArea.objectAttributes, TPMA_OBJECT, fixedTPM)))
    {
        // Do the cryptographic key validation
        result =
            CryptValidateKeys(publicArea, sensitive, blamePublic, blameSensitive);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
#if ALG_RSA
    // If this is an RSA key, then expand the private exponent.
    // Note: ObjectLoad() is only called by TPM2_Import() if the parent is fixedTPM.
    // For any key that does not have a fixedTPM parent, the exponent is computed
    // whenever it is loaded
    if((publicArea->type == TPM_ALG_RSA) && (sensitive != NULL))
    {
        result = CryptRsaLoadPrivateExponent(publicArea, sensitive);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
#endif  // ALG_RSA
    // See if there is an object to populate
    if((result == TPM_RC_SUCCESS) && (object != NULL))
    {
        // Initialize public
        object->publicArea = *publicArea;
        // Copy sensitive if there is one
        if(sensitive == NULL)
            object->attributes.publicOnly = SET;
        else
            object->sensitive = *sensitive;
        // Set the name, if one was provided
```

```
        if(name != NULL)
            object->name = *name;
        else
            object->name.t.size = 0;
    }
    return result;
}

//*** AllocateSequenceSlot()
// This function allocates a sequence slot and initializes the parts that
// are used by the normal objects so that a sequence object is not inadvertently
// used for an operation that is not appropriate for a sequence.
//
static HASH_OBJECT* AllocateSequenceSlot(
    TPM_HANDLE* newHandle,  // OUT: receives the allocated handle
    TPM2B_AUTH* auth        // IN: the authValue for the slot
)
{
    HASH_OBJECT* object = (HASH_OBJECT*)ObjectAllocateSlot(newHandle);
    //
    // Validate that the proper location of the hash state data relative to the
    // object state data. It would be good if this could have been done at compile
    // time but it can't so do it in something that can be removed after debug.
    MUST_BE(offsetof(HASH_OBJECT, auth) == offsetof(OBJECT, publicArea.authPolicy));

    if(object != NULL)
    {

        // Set the common values that a sequence object shares with an ordinary object
        // First, clear all attributes
        MemorySet(&object->objectAttributes, 0, sizeof(TPMA_OBJECT));

        // The type is TPM_ALG_NULL
        object->type = TPM_ALG_NULL;

        // This has no name algorithm and the name is the Empty Buffer
        object->nameAlg = TPM_ALG_NULL;

        // A sequence object is considered to be in the NULL hierarchy so it should
        // be marked as temporary so that it can't be persisted
        object->attributes.temporary = SET;

        // A sequence object is DA exempt.
        SET_ATTRIBUTE(object->objectAttributes, TPMA_OBJECT, noDA);

        // Copy the authorization value
        if(auth != NULL)
            object->auth = *auth;
        else
            object->auth.t.size = 0;
    }
    return object;
}

#if CC_HMAC_Start || CC_MAC_Start
//*** ObjectCreateHMACSequence()
// This function creates an internal HMAC sequence object.
//  Return Type: TPM_RC
//      TPM_RC_OBJECT_MEMORY        if there is no free slot for an object
TPM_RC
ObjectCreateHMACSequence(
    TPMI_ALG_HASH   hashAlg,    // IN: hash algorithm
    OBJECT*         keyObject,  // IN: the object containing the HMAC key
    TPM2B_AUTH*     auth,       // IN: authValue
    TPMI_DH_OBJECT* newHandle   // OUT: HMAC sequence object handle
)
```

```
{
    HASH_OBJECT* hmacObject;
    //
    // Try to allocate a slot for new object
    hmacObject = AllocateSequenceSlot(newHandle, auth);

    if(hmacObject == NULL)
        return TPM_RC_OBJECT_MEMORY;
    // Set HMAC sequence bit
    hmacObject->attributes.hmacSeq = SET;

#  if !SMAC_IMPLEMENTED
    if(CryptHmacStart(&hmacObject->state.hmacState,
                      hashAlg,
                      keyObject->sensitive.sensitive.bits.b.size,
                      keyObject->sensitive.sensitive.bits.b.buffer)
       == 0)
#  else
    if(CryptMacStart(&hmacObject->state.hmacState,
                     &keyObject->publicArea.parameters,
                     hashAlg,
                     &keyObject->sensitive.sensitive.any.b)
       == 0)
#  endif  // SMAC_IMPLEMENTED
        return TPM_RC_FAILURE;
    return TPM_RC_SUCCESS;
}
#endif

//*** ObjectCreateHashSequence()
// This function creates a hash sequence object.
//  Return Type: TPM_RC
//      TPM_RC_OBJECT_MEMORY        if there is no free slot for an object
TPM_RC
ObjectCreateHashSequence(TPMI_ALG_HASH   hashAlg,   // IN: hash algorithm
                         TPM2B_AUTH*     auth,      // IN: authValue
                         TPMI_DH_OBJECT* newHandle  // OUT: sequence object handle
)
{
    HASH_OBJECT* hashObject = AllocateSequenceSlot(newHandle, auth);
    //
    // See if slot allocated
    if(hashObject == NULL)
        return TPM_RC_OBJECT_MEMORY;
    // Set hash sequence bit
    hashObject->attributes.hashSeq = SET;

    // Start hash for hash sequence
    CryptHashStart(&hashObject->state.hashState[0], hashAlg);

    return TPM_RC_SUCCESS;
}

//*** ObjectCreateEventSequence()
// This function creates an event sequence object.
//  Return Type: TPM_RC
//      TPM_RC_OBJECT_MEMORY        if there is no free slot for an object
TPM_RC
ObjectCreateEventSequence(TPM2B_AUTH*     auth,      // IN: authValue
                          TPMI_DH_OBJECT* newHandle  // OUT: sequence object handle
)
{
    HASH_OBJECT* hashObject = AllocateSequenceSlot(newHandle, auth);
    UINT32       count;
    TPM_ALG_ID   hash;
    //
```

```
    // See if slot allocated
    if(hashObject == NULL)
        return TPM_RC_OBJECT_MEMORY;
    // Set the event sequence attribute
    hashObject->attributes.eventSeq = SET;

    // Initialize hash states for each implemented PCR algorithms
    for(count = 0; (hash = CryptHashGetAlgByIndex(count)) != TPM_ALG_NULL; count++)
        CryptHashStart(&hashObject->state.hashState[count], hash);
    return TPM_RC_SUCCESS;
}

//*** ObjectTerminateEvent()
// This function is called to close out the event sequence and clean up the hash
// context states.
void ObjectTerminateEvent(void)
{
    HASH_OBJECT* hashObject;
    int          count;
    BYTE         buffer[MAX_DIGEST_SIZE];
    //
    hashObject = (HASH_OBJECT*)HandleToObject(g_DRTMHandle);

    // Don't assume that this is a proper sequence object
    if(hashObject->attributes.eventSeq)
    {
        // If it is, close any open hash contexts. This is done in case
        // the cryptographic implementation has some context values that need to be
        // cleaned up (hygiene).
        //
        for(count = 0; CryptHashGetAlgByIndex(count) != TPM_ALG_NULL; count++)
        {
            CryptHashEnd(&hashObject->state.hashState[count], 0, buffer);
        }
        // Flush sequence object
        FlushObject(g_DRTMHandle);
    }
    g_DRTMHandle = TPM_RH_UNASSIGNED;
}

//*** ObjectContextLoad()
// This function loads an object from a saved object context.
//  Return Type: OBJECT *
//      NULL        if there is no free slot for an object
//      != NULL     points to the loaded object
OBJECT* ObjectContextLoad(
    ANY_OBJECT_BUFFER* object,  // IN: pointer to object structure in saved
                                //     context
    TPMI_DH_OBJECT* handle      // OUT: object handle
)
{
    OBJECT* newObject = ObjectAllocateSlot(handle);
    //
    // Try to allocate a slot for new object
    if(newObject != NULL)
    {
        // Copy the first part of the object
        MemoryCopy(newObject, object, offsetof(HASH_OBJECT, state));
        // See if this is a sequence object
        if(ObjectIsSequence(newObject))
        {
            // If this is a sequence object, import the data
            SequenceDataImport((HASH_OBJECT*)newObject, (HASH_OBJECT_BUFFER*)object);
        }
        else
        {
```

```
            // Copy input object data to internal structure
            MemoryCopy(newObject, object, sizeof(OBJECT));
        }
    }
    return newObject;
}

//*** FlushObject()
// This function frees an object slot.
//
// This function requires that the object is loaded.
void FlushObject(TPMI_DH_OBJECT handle  // IN: handle to be freed
)
{
    UINT32 index = handle - TRANSIENT_FIRST;
    //
    pAssert(index < MAX_LOADED_OBJECTS);
    // Clear all the object attributes
    MemorySet((BYTE*)&(s_objects[index].attributes), 0, sizeof(OBJECT_ATTRIBUTES));
    return;
}

//*** ObjectFlushHierarchy()
// This function is called to flush all the loaded transient objects associated
// with a hierarchy when the hierarchy is disabled.
void ObjectFlushHierarchy(TPMI_RH_HIERARCHY hierarchy  // IN: hierarchy to be flush
)
{
    UINT16 i;
    //
    // iterate object slots
    for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    {
        if(s_objects[i].attributes.occupied)  // If found an occupied slot
        {
            switch(hierarchy)
            {
                case TPM_RH_PLATFORM:
                    if(s_objects[i].attributes.ppsHierarchy == SET)
                        s_objects[i].attributes.occupied = FALSE;
                    break;
                case TPM_RH_OWNER:
                    if(s_objects[i].attributes.spsHierarchy == SET)
                        s_objects[i].attributes.occupied = FALSE;
                    break;
                case TPM_RH_ENDORSEMENT:
                    if(s_objects[i].attributes.epsHierarchy == SET)
                        s_objects[i].attributes.occupied = FALSE;
                    break;
                default:
                    FAIL(FATAL_ERROR_INTERNAL);
                    break;
            }
        }
    }

    return;
}

//*** ObjectLoadEvict()
// This function loads a persistent object into a transient object slot.
//
// This function requires that 'handle' is associated with a persistent object.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE               the persistent object does not exist
//                                  or the associated hierarchy is disabled.
```

```
//      TPM_RC_OBJECT_MEMORY        no object slot
TPM_RC
ObjectLoadEvict(TPM_HANDLE* handle,  // IN:OUT: evict object handle.  If success, it
                                     // will be replace by the loaded object handle
                COMMAND_INDEX commandIndex  // IN: the command being processed
)
{
    TPM_RC     result;
    TPM_HANDLE evictHandle = *handle;  // Save the evict handle
    OBJECT*    object;
    //
    // If this is an index that references a persistent object created by
    // the platform, then return TPM_RH_HANDLE if the phEnable is FALSE
    if(*handle >= PLATFORM_PERSISTENT)
    {
        // belongs to platform
        if(g_phEnable == CLEAR)
            return TPM_RC_HANDLE;
    }
    // belongs to owner
    else if(gc.shEnable == CLEAR)
        return TPM_RC_HANDLE;
    // Try to allocate a slot for an object
    object = ObjectAllocateSlot(handle);
    if(object == NULL)
        return TPM_RC_OBJECT_MEMORY;
    // Copy persistent object to transient object slot.  A TPM_RC_HANDLE
    // may be returned at this point. This will mark the slot as containing
    // a transient object so that it will be flushed at the end of the
    // command
    result = NvGetEvictObject(evictHandle, object);

    // Bail out if this failed
    if(result != TPM_RC_SUCCESS)
        return result;
    // check the object to see if it is in the endorsement hierarchy
    // if it is and this is not a TPM2_EvictControl() command, indicate
    // that the hierarchy is disabled.
    // If the associated hierarchy is disabled, make it look like the
    // handle is not defined
    if(HierarchyNormalizeHandle(object->hierarchy) == TPM_RH_ENDORSEMENT
       && gc.ehEnable == CLEAR && GetCommandCode(commandIndex) != TPM_CC_EvictControl)
        return TPM_RC_HANDLE;

    return result;
}

//*** ObjectComputeName()
// This does the name computation from a public area (can be marshaled or not).
TPM2B_NAME* ObjectComputeName(UINT32     size,  // IN: the size of the area to digest
                              BYTE*      publicArea,  // IN: the public area to digest
                              TPM_ALG_ID nameAlg,     // IN: the hash algorithm to use
                              TPM2B_NAME* name        // OUT: Computed name
)
{
    // Hash the publicArea into the name buffer leaving room for the nameAlg
    name->t.size = CryptHashBlock(
        nameAlg, size, publicArea, sizeof(name->t.name) - 2, &name->t.name[2]);
    // set the nameAlg
    UINT16_TO_BYTE_ARRAY(nameAlg, name->t.name);
    name->t.size += 2;
    return name;
}

//*** PublicMarshalAndComputeName()
// This function computes the Name of an object from its public area.
```

```
TPM2B_NAME* PublicMarshalAndComputeName(
    TPMT_PUBLIC* publicArea,  // IN: public area of an object
    TPM2B_NAME*  name         // OUT: name of the object
)
{
    // Will marshal a public area into a template. This is because the internal
    // format for a TPM2B_PUBLIC is a structure and not a simple BYTE buffer.
    TPM2B_TEMPLATE marshaled;  // this is big enough to hold a
                               //  marshaled TPMT_PUBLIC
    BYTE* buffer = (BYTE*)&marshaled.t.buffer;
    //
    // if the nameAlg is NULL then there is no name.
    if(publicArea->nameAlg == TPM_ALG_NULL)
        name->t.size = 0;
    else
    {
        // Marshal the public area into its canonical form
        marshaled.t.size = TPMT_PUBLIC_Marshal(publicArea, &buffer, NULL);
        // and compute the name
        ObjectComputeName(
            marshaled.t.size, marshaled.t.buffer, publicArea->nameAlg, name);
    }
    return name;
}

//*** ComputeQualifiedName()
// This function computes the qualified name of an object.
void ComputeQualifiedName(
    TPM_HANDLE  parentHandle,  // IN: parent's handle
    TPM_ALG_ID  nameAlg,       // IN: name hash
    TPM2B_NAME* name,          // IN: name of the object
    TPM2B_NAME* qualifiedName  // OUT: qualified name of the object
)
{
    HASH_STATE hashState;  // hash state
    TPM2B_NAME parentName;
    //
    if(parentHandle == TPM_RH_UNASSIGNED)
    {
        MemoryCopy2B(&qualifiedName->b, &name->b, sizeof(qualifiedName->t.name));
        *qualifiedName = *name;
    }
    else
    {
        GetQualifiedName(parentHandle, &parentName);

        //      QN_A = hash_A (QN of parent || NAME_A)

        // Start hash
        qualifiedName->t.size = CryptHashStart(&hashState, nameAlg);

        // Add parent's qualified name
        CryptDigestUpdate2B(&hashState, &parentName.b);

        // Add self name
        CryptDigestUpdate2B(&hashState, &name->b);

        // Complete hash leaving room for the name algorithm
        CryptHashEnd(&hashState, qualifiedName->t.size, &qualifiedName->t.name[2]);
        UINT16_TO_BYTE_ARRAY(nameAlg, qualifiedName->t.name);
        qualifiedName->t.size += 2;
    }
    return;
}

//*** ObjectIsStorage()
```

```
// This function determines if an object has the attributes associated
// with a parent. A parent is an asymmetric or symmetric block cipher key
// that has its 'restricted' and 'decrypt' attributes SET, and 'sign' CLEAR.
//  Return Type: BOOL
//      TRUE(1)         object is a storage key
//      FALSE(0)        object is not a storage key
BOOL ObjectIsStorage(TPMI_DH_OBJECT handle  // IN: object handle
)
{
    OBJECT*      object     = HandleToObject(handle);
    TPMT_PUBLIC* publicArea = ((object != NULL) ? &object->publicArea : NULL);
    //
    return (publicArea != NULL
            && IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted)
            && IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, decrypt)
            && !IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign)
            && (object->publicArea.type == TPM_ALG_RSA
                || object->publicArea.type == TPM_ALG_ECC));
}

//*** ObjectCapGetLoaded()
// This function returns a list of handles of loaded object, starting from
// 'handle'. 'Handle' must be in the range of valid transient object handles,
// but does not have to be the handle of a loaded transient object.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
ObjectCapGetLoaded(TPMI_DH_OBJECT handle,     // IN: start handle
                   UINT32         count,      // IN: count of returned handles
                   TPML_HANDLE*   handleList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;
    //
    pAssert(HandleGetType(handle) == TPM_HT_TRANSIENT);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES)
        count = MAX_CAP_HANDLES;

    // Iterate object slots to get loaded object handles
    for(i = handle - TRANSIENT_FIRST; i < MAX_LOADED_OBJECTS; i++)
    {
        if(s_objects[i].attributes.occupied == TRUE)
        {
            // A valid transient object can not be the copy of a persistent object
            pAssert(s_objects[i].attributes.evict == CLEAR);

            if(handleList->count < count)
            {
                // If we have not filled up the return list, add this object
                // handle to it
                handleList->handle[handleList->count] = i + TRANSIENT_FIRST;
                handleList->count++;
            }
            else
            {
                // If the return list is full but we still have loaded object
                // available, report this and stop iterating
                more = YES;
                break;
```

```
            }
        }
    }

    return more;
}

//*** ObjectCapGetOneLoaded()
// This function returns whether a handle is loaded.
BOOL ObjectCapGetOneLoaded(TPMI_DH_OBJECT handle)  // IN: handle
{
    UINT32 i;

    pAssert(HandleGetType(handle) == TPM_HT_TRANSIENT);

    // Iterate object slots to get loaded object handles
    for(i = handle - TRANSIENT_FIRST; i < MAX_LOADED_OBJECTS; i++)
    {
        if(s_objects[i].attributes.occupied == TRUE)
        {
            // A valid transient object can not be the copy of a persistent object
            pAssert(s_objects[i].attributes.evict == CLEAR);

            return TRUE;
        }
    }

    return FALSE;
}

//*** ObjectCapGetTransientAvail()
// This function returns an estimate of the number of additional transient
// objects that could be loaded into the TPM.
UINT32
ObjectCapGetTransientAvail(void)
{
    UINT32 i;
    UINT32 num = 0;
    //
    // Iterate object slot to get the number of unoccupied slots
    for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    {
        if(s_objects[i].attributes.occupied == FALSE)
            num++;
    }

    return num;
}

//*** ObjectGetPublicAttributes()
// Returns the attributes associated with an object handles.
TPMA_OBJECT
ObjectGetPublicAttributes(TPM_HANDLE handle)
{
    return HandleToObject(handle)->publicArea.objectAttributes;
}

OBJECT_ATTRIBUTES
ObjectGetProperties(TPM_HANDLE handle)
{
    return HandleToObject(handle)->attributes;
}
```

7.176 /tpm/src/subsystem/PCR.c

```
//** Introduction
//
// This function contains the functions needed for PCR access and manipulation.
//
// This implementation uses a static allocation for the PCR. The amount of
// memory is allocated based on the number of PCR in the implementation and
// the number of implemented hash algorithms. This is not the expected
// implementation. PCR SPACE DEFINITIONS.
//
// In the definitions below, the g_hashPcrMap is a bit array that indicates
// which of the PCR are implemented. The g_hashPcr array is an array of digests.
// In this implementation, the space is allocated whether the PCR is implemented
// or not.

//** Includes, Defines, and Data Definitions
#define PCR_C
#include "Tpm.h"

// verify values from pcrstruct.h. not <= because group #0 is reserved
// indicating no auth/policy support
TPM_STATIC_ASSERT(NUM_AUTHVALUE_PCR_GROUP < (1 << MAX_PCR_GROUP_BITS));
TPM_STATIC_ASSERT(NUM_POLICY_PCR_GROUP < (1 << MAX_PCR_GROUP_BITS));

//** Functions

//*** PCRBelongsAuthGroup()
// This function indicates if a PCR belongs to a group that requires an authValue
// in order to modify the PCR.  If it does, 'groupIndex' is set to value of
// the group index.  This feature of PCR is decided by the platform specification.
//
//  Return Type: BOOL
//      TRUE(1)         PCR belongs an authorization group
//      FALSE(0)        PCR does not belong an authorization group
BOOL PCRBelongsAuthGroup(TPMI_DH_PCR handle,     // IN: handle of PCR
                         UINT32*     groupIndex  // OUT: group array index if PCR
                         //      belongs to a group that allows authValue.  If PCR
                         //      does not belong to an authorization
                         //      group, the value in this parameter is zero
)
{
    *groupIndex = 0;

#if defined NUM_AUTHVALUE_PCR_GROUP && NUM_AUTHVALUE_PCR_GROUP > 0
    // Platform specification determines to which authorization group a PCR belongs
    // (if any). In this implementation, we assume there is only
    // one authorization group which contains PCR[20-22].  If the platform
    // specification requires differently, the implementation should be changed
    // accordingly
    UINT32         pcr = handle - PCR_FIRST;
    PCR_Attributes currentPcrAttributes =
        _platPcr__GetPcrInitializationAttributes(pcr);

    if(currentPcrAttributes.authValuesGroup != 0)
    {
        // turn 1-based group number into actual array index expected by callers
        *groupIndex = currentPcrAttributes.authValuesGroup - 1;
        pAssert_BOOL(*groupIndex < NUM_AUTHVALUE_PCR_GROUP);
        return TRUE;
    }

#endif
    return FALSE;
}
```

```
//*** PCRBelongsPolicyGroup()
// This function indicates if a PCR belongs to a group that requires a policy
// authorization in order to modify the PCR.  If it does, 'groupIndex' is set
// to value of the group index.  This feature of PCR is decided by the platform
// specification.
// return type: BOOL
//      TRUE:           PCR belongs a policy group
//      FALSE:          PCR does not belong a policy group
BOOL PCRBelongsPolicyGroup(
    TPMI_DH_PCR handle,     // IN: handle of PCR
    UINT32*     groupIndex  // OUT: group index if PCR belongs a group that
                            //     allows policy.  If PCR does not belong to
                            //     a policy group, the value in this
                            //     parameter is zero
)
{
    *groupIndex = 0;

#if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
    // Platform specification decides if a PCR belongs to a policy group and
    // belongs to which group.
    UINT32         pcr = handle - PCR_FIRST;
    PCR_Attributes currentPcrAttributes =
        _platPcr__GetPcrInitializationAttributes(pcr);
    if(currentPcrAttributes.policyAuthGroup != 0)
    {
        // turn 1-based group number into actual array index expected by callers
        *groupIndex = currentPcrAttributes.policyAuthGroup - 1;
        pAssert_BOOL(*groupIndex < NUM_POLICY_PCR_GROUP);
        return TRUE;
    }
#endif
    return FALSE;
}

//*** PCRBelongsTCBGroup()
// This function indicates if a PCR belongs to the TCB group.
// return type: BOOL
//      TRUE:           PCR belongs to TCB group
//      FALSE:          PCR does not belong to TCB group
static BOOL PCRBelongsTCBGroup(TPMI_DH_PCR handle  // IN: handle of PCR
)
{
#if ENABLE_PCR_NO_INCREMENT == YES
    // Platform specification decides if a PCR belongs to a TCB group.
    UINT32         pcr = handle - PCR_FIRST;
    PCR_Attributes currentPcrAttributes =
        _platPcr__GetPcrInitializationAttributes(pcr);
    return currentPcrAttributes.doNotIncrementPcrCounter;
#else
    return FALSE;
#endif
}

//*** PCRPolicyIsAvailable()
// This function indicates if a policy is available for a PCR.
// return type: BOOL
//      TRUE        the PCR may be authorized by policy
//      FALSE       the PCR does not allow policy
BOOL PCRPolicyIsAvailable(TPMI_DH_PCR handle  // IN: PCR handle
)
{
    UINT32 groupIndex;

    return PCRBelongsPolicyGroup(handle, &groupIndex);
}
```

```
//*** PCRGetAuthValue()
// This function is used to access the authValue of a PCR.  If PCR does not
// belong to an authValue group, an EmptyAuth will be returned.
TPM2B_AUTH* PCRGetAuthValue(TPMI_DH_PCR handle  // IN: PCR handle
)
{
    UINT32 groupIndex;

    if(PCRBelongsAuthGroup(handle, &groupIndex))
    {
        return &gc.pcrAuthValues.auth[groupIndex];
    }
    else
    {
        return NULL;
    }
}

//*** PCRGetAuthPolicy()
// This function is used to access the authorization policy of a PCR. It sets
// 'policy' to the authorization policy and returns the hash algorithm for policy
//  If the PCR does not allow a policy, TPM_ALG_NULL is returned.
TPMI_ALG_HASH
PCRGetAuthPolicy(TPMI_DH_PCR   handle,  // IN: PCR handle
                 TPM2B_DIGEST* policy   // OUT: policy of PCR
)
{
    UINT32 groupIndex;

    if(PCRBelongsPolicyGroup(handle, &groupIndex))
    {
        *policy = gp.pcrPolicies.policy[groupIndex];
        return gp.pcrPolicies.hashAlg[groupIndex];
    }
    else
    {
        policy->t.size = 0;
        return TPM_ALG_NULL;
    }
}

//*** PCRManufacture()
// This function is used to initialize the policies when a TPM is manufactured.
// This function would only be called in a manufacturing environment or in
// a TPM simulator.
void PCRManufacture(void)
{
    UINT32 i;
#if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
    for(i = 0; i < NUM_POLICY_PCR_GROUP; i++)
    {
        gp.pcrPolicies.hashAlg[i]       = TPM_ALG_NULL;
        gp.pcrPolicies.policy[i].t.size = 0;
    }
#endif
#if defined NUM_AUTHVALUE_PCR_GROUP && NUM_AUTHVALUE_PCR_GROUP > 0
    for(i = 0; i < NUM_AUTHVALUE_PCR_GROUP; i++)
    {
        gc.pcrAuthValues.auth[i].t.size = 0;
    }
#endif
    // We need to give an initial configuration on allocated PCR before
    // receiving any TPM2_PCR_Allocate command to change this configuration
    // When the simulation environment starts, we allocate all the PCRs
    for(gp.pcrAllocated.count = 0; gp.pcrAllocated.count < HASH_COUNT;
```

```
        gp.pcrAllocated.count++)
    {
        TPM_ALG_ID currentBank  = CryptHashGetAlgByIndex(gp.pcrAllocated.count);
        BOOL       isBankActive = _platPcr_IsPcrBankDefaultActive(currentBank);

        gp.pcrAllocated.pcrSelections[gp.pcrAllocated.count].hash = currentBank;

        gp.pcrAllocated.pcrSelections[gp.pcrAllocated.count].sizeofSelect =
            PCR_SELECT_MAX;
        for(i = 0; i < PCR_SELECT_MAX; i++)
        {
            gp.pcrAllocated.pcrSelections[gp.pcrAllocated.count].pcrSelect[i] =
                isBankActive ? 0xFF : 0;
        }
    }

    // Store the initial configuration to NV
    NV_SYNC_PERSISTENT(pcrPolicies);
    NV_SYNC_PERSISTENT(pcrAllocated);

    return;
}

//*** GetSavedPcrPointer()
// This function returns the address of an array of state saved PCR based
// on the hash algorithm.
//
//  Return Type: BYTE *
//      NULL            no such algorithm
//      != NULL         pointer to the 0th byte of the 0th PCR
static BYTE* GetSavedPcrPointer(TPM_ALG_ID alg,      // IN: algorithm for bank
                                UINT32     pcrIndex  // IN: PCR index in PCR_SAVE
)
{
    BYTE* retVal = NULL;
    switch(alg)
    {
#define HASH_CASE(HASH, Hash)                \
    case TPM_ALG_##HASH:                     \
        retVal = gc.pcrSave.Hash[pcrIndex]; \
        break;

        FOR_EACH_HASH(HASH_CASE)
#undef HASH_CASE

        default:
            FAIL_NULL(FATAL_ERROR_INTERNAL);
    }
    return retVal;
}

//*** PcrIsAllocated()
// This function indicates if a PCR number for the particular hash algorithm
// is allocated.
//  Return Type: BOOL
//      TRUE(1)         PCR is allocated
//      FALSE(0)        PCR is not allocated
BOOL PcrIsAllocated(UINT32        pcr,     // IN: The number of the PCR
                    TPMI_ALG_HASH hashAlg  // IN: The PCR algorithm
)
{
    UINT32 i;
    BOOL   allocated = FALSE;

    if(pcr < IMPLEMENTATION_PCR)
    {
```

```
        for(i = 0; i < gp.pcrAllocated.count; i++)
        {
            if(gp.pcrAllocated.pcrSelections[i].hash == hashAlg)
            {
                if(((gp.pcrAllocated.pcrSelections[i].pcrSelect[pcr / 8])
                    & (1 << (pcr % 8)))
                   != 0)
                    allocated = TRUE;
                else
                    allocated = FALSE;
                break;
            }
        }
    }
    return allocated;
}

// Get pointer to particular PCR from bank (array)
// CAUTION: This function does not validate the pcrNumber
// vs the size of the array.
// See Also: GetPcrPointerIfAllocated
static BYTE* GetPcrPointerFromPcrArray(PCR*       pPcrArray,
                                       TPM_ALG_ID alg,       // IN: algorithm for bank
                                       UINT32     pcrNumber  // IN: PCR number
)
{
    switch(alg)
    {
#if ALG_SHA1
        case TPM_ALG_SHA1:
            return pPcrArray[pcrNumber].Sha1Pcr;
#endif
#if ALG_SHA256
        case TPM_ALG_SHA256:
            return pPcrArray[pcrNumber].Sha256Pcr;
#endif
#if ALG_SHA384
        case TPM_ALG_SHA384:
            return pPcrArray[pcrNumber].Sha384;
#endif
#if ALG_SHA512
        case TPM_ALG_SHA512:
            return pPcrArray[pcrNumber].Sha512;
#endif
#if ALG_SM3_256
        case TPM_ALG_SM3_256:
            return pPcrArray[pcrNumber].Sm3_256;
#endif
#if ALG_SHA3_256
        case TPM_ALG_SHA3_256:
            return pPcrArray[pcrNumber].Sha3_256;
#endif
#if ALG_SHA3_384
        case TPM_ALG_SHA3_384:
            return pPcrArray[pcrNumber].Sha3_384;
#endif
#if ALG_SHA3_512
        case TPM_ALG_SHA3_512:
            return pPcrArray[pcrNumber].Sha3_512;
#endif
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    return NULL;
}
```

```
BYTE* GetPcrPointerIfAllocated(PCR*       pPcrArray,
                               TPM_ALG_ID alg,       // IN: algorithm for bank
                               UINT32     pcrNumber  // IN: PCR number
)
{
    //
    if(!PcrIsAllocated(pcrNumber, alg))
        return NULL;

    return GetPcrPointerFromPcrArray(pPcrArray,
                                     alg,       // IN: algorithm for bank
                                     pcrNumber  // IN: PCR number
    );
}

//*** GetPcrPointer()
// This function returns the address of an array of PCR based on the
// hash algorithm.
//
//  Return Type: BYTE *
//      NULL            no such algorithm
//      != NULL         pointer to the 0th byte of the requested PCR
BYTE* GetPcrPointer(TPM_ALG_ID alg,       // IN: algorithm for bank
                    UINT32     pcrNumber  // IN: PCR number
)
{
    return GetPcrPointerIfAllocated(s_pcrs, alg, pcrNumber);
}

//*** IsPcrSelected()
// This function indicates if an indicated PCR number is selected by the bit map in
// 'selection'.
//
//  Return Type: BOOL
//      TRUE(1)         PCR is selected
//      FALSE(0)        PCR is not selected
static BOOL IsPcrSelected(
    UINT32              pcr,       // IN: The number of the PCR
    TPMS_PCR_SELECTION* selection  // IN: The selection structure
)
{
    BOOL selected;
    selected = (pcr < IMPLEMENTATION_PCR
                && ((selection->pcrSelect[pcr / 8]) & (1 << (pcr % 8))) != 0);
    return selected;
}

//*** FilterPcr()
// This function modifies a PCR selection array based on the implemented
// PCR.
static void FilterPcr(TPMS_PCR_SELECTION* selection  // IN: input PCR selection
)
{
    UINT32              i;
    TPMS_PCR_SELECTION* allocated = NULL;

    // If size of select is less than PCR_SELECT_MAX, zero the unspecified PCR
    for(i = selection->sizeofSelect; i < PCR_SELECT_MAX; i++)
        selection->pcrSelect[i] = 0;

    // Find the internal configuration for the bank
    for(i = 0; i < gp.pcrAllocated.count; i++)
    {
        if(gp.pcrAllocated.pcrSelections[i].hash == selection->hash)
        {
```

```
            allocated = &gp.pcrAllocated.pcrSelections[i];
            break;
        }
    }

    for(i = 0; i < selection->sizeofSelect; i++)
    {
        if(allocated == NULL)
        {
            // If the required bank does not exist, clear input selection
            selection->pcrSelect[i] = 0;
        }
        else
            selection->pcrSelect[i] &= allocated->pcrSelect[i];
    }

    return;
}

//*** PcrDrtm()
// This function does the DRTM and H-CRTM processing it is called from
// _TPM_Hash_End.
void PcrDrtm(const TPMI_DH_PCR pcrHandle,  // IN: the index of the PCR to be
                                           //     modified
             const TPMI_ALG_HASH hash,     // IN: the bank identifier
             const TPM2B_DIGEST* digest    // IN: the digest to modify the PCR
)
{
    BYTE* pcrData = GetPcrPointer(hash, pcrHandle);

    if(pcrData != NULL)
    {
        // Rest the PCR to zeros
        MemorySet(pcrData, 0, digest->t.size);

        // if the TPM has not started, then set the PCR to 0...04 and then extend
        if(!TPMIsStarted())
        {
            pcrData[digest->t.size - 1] = 4;
        }
        // Now, extend the value
        PCRExtend(pcrHandle, hash, digest->t.size, (BYTE*)digest->t.buffer);
    }
}

//*** PCR_ClearAuth()
// This function is used to reset the PCR authorization values. It is called
// on TPM2_Startup(CLEAR) and TPM2_Clear().
void PCR_ClearAuth(void)
{
#if defined NUM_AUTHVALUE_PCR_GROUP && NUM_AUTHVALUE_PCR_GROUP > 0
    int j;
    for(j = 0; j < NUM_AUTHVALUE_PCR_GROUP; j++)
    {
        gc.pcrAuthValues.auth[j].t.size = 0;
    }
#endif
}

//*** PCRStartup()
// This function initializes the PCR subsystem at TPM2_Startup().
BOOL PCRStartup(STARTUP_TYPE type,     // IN: startup type
                BYTE         locality  // IN: startup locality
)
{
    UINT32 pcr, j;
```

```
    UINT32 saveIndex = 0;

    g_pcrReConfig    = FALSE;

    // Don't test for SU_RESET because that should be the default when nothing
    // else is selected
    if(type != SU_RESUME && type != SU_RESTART)
    {
        // PCR generation counter is cleared at TPM_RESET
        gr.pcrCounter = 0;
    }

    // check the TPM library and platform are properly paired.
    // if this fails the platform and library are compiled with different
    // definitions of the number of PCRs - immediately enter FAILURE mode and
    // return FALSE
    pAssert_BOOL(_platPcr__NumberOfPcrs() == IMPLEMENTATION_PCR);

    // Initialize/Restore PCR values
    for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        // On resume, need to know if this PCR had its state saved or not
        UINT32 stateSaved;
        // note structure is a bitfield and returned by value.
        PCR_Attributes currentPcrAttributes =
            _platPcr__GetPcrInitializationAttributes(pcr);

        if(type == SU_RESUME && currentPcrAttributes.stateSave == SET)
        {
            stateSaved = 1;
        }
        else
        {
            stateSaved = 0;
            PCRChanged(pcr);
        }

        // If this is the H-CRTM PCR and we are not doing a resume and we
        // had an H-CRTM event, then we don't change this PCR
        if(pcr == HCRTM_PCR && type != SU_RESUME && g_DrtmPreStartup == TRUE)
            continue;

        // Iterate each hash algorithm bank
        for(j = 0; j < gp.pcrAllocated.count; j++)
        {
            TPMI_ALG_HASH hash    = gp.pcrAllocated.pcrSelections[j].hash;
            BYTE*         pcrData = GetPcrPointer(hash, pcr);
            UINT16        pcrSize = CryptHashGetDigestSize(hash);

            if(pcrData != NULL)
            {
                // if state was saved
                if(stateSaved == 1)
                {
                    // Restore saved PCR value
                    BYTE* pcrSavedData;
                    pcrSavedData = GetSavedPcrPointer(hash, saveIndex);
                    if(pcrSavedData == NULL)
                        return FALSE;
                    MemoryCopy(pcrData, pcrSavedData, pcrSize);
                }
                else  // PCR was not restored by state save
                {
                    // give platform opportunity to provide the PCR initialization
                    // value and it's length. this provides a platform specification
                    // the ability to change the default values without affecting the
```

```
                    // core library. if the platform doesn't have a value, then the
                    // result is expected to be TPM_RC_PCR and the size to be 0 and we
                    // provide the original defaults.
                    uint16_t pcrLength        = 0;
                    TPM_RC   pcrInitialResult = _platPcr__GetInitialValueForPcr(
                        pcr, hash, locality, pcrData, pcrSize, &pcrLength);

                    // any other result is a fatal error
                    pAssert_BOOL(pcrInitialResult == TPM_RC_SUCCESS
                                 || pcrInitialResult == TPM_RC_PCR);
                    if(pcrInitialResult == TPM_RC_SUCCESS && pcrLength == pcrSize)
                    {
                        // just use the PCR initialized by platform
                    }
                    else
                    {
                        // If the reset locality contains locality 4, then this
                        // indicates a DRTM PCR where the reset value is all ones,
                        // otherwise it is all zero.  Don't check with equal because
                        // resetLocality is a bitfield of multiple values and does
                        // not support extended localities.
                        BYTE defaultValue = 0;
                        if((currentPcrAttributes.resetLocality & 0x10) != 0)
                        {
                            defaultValue = 0xFF;
                        }
                        MemorySet(pcrData, defaultValue, pcrSize);
                        if(pcr == HCRTM_PCR)
                        {
                            pcrData[pcrSize - 1] = locality;
                        }
                    }
                }
            }
        }
        saveIndex += stateSaved;
    }
    // Reset authValues on TPM2_Startup(CLEAR)
    if(type != SU_RESUME)
        PCR_ClearAuth();
    return TRUE;
}

//*** PCRStateSave()
// This function is used to save the PCR values that will be restored on TPM Resume.
void PCRStateSave(TPM_SU type  // IN: startup type
)
{
    UINT32 pcr, j;
    UINT32 saveIndex = 0;

    // if state save CLEAR, nothing to be done.  Return here
    if(type == TPM_SU_CLEAR)
        return;

    // Copy PCR values to the structure that should be saved to NV
    for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        PCR_Attributes currentPcrAttributes =
            _platPcr__GetPcrInitializationAttributes(pcr);

        UINT32 stateSaved = (currentPcrAttributes.stateSave == SET) ? 1 : 0;

        // Iterate each hash algorithm bank
        for(j = 0; j < gp.pcrAllocated.count; j++)
        {
```

```
            BYTE*  pcrData;
            UINT32 pcrSize;

            pcrData = GetPcrPointer(gp.pcrAllocated.pcrSelections[j].hash, pcr);

            if(pcrData != NULL)
            {
                pcrSize =
                    CryptHashGetDigestSize(gp.pcrAllocated.pcrSelections[j].hash);

                if(stateSaved == 1)
                {
                    // Restore saved PCR value
                    BYTE* pcrSavedData;
                    pcrSavedData = GetSavedPcrPointer(
                        gp.pcrAllocated.pcrSelections[j].hash, saveIndex);
                    MemoryCopy(pcrSavedData, pcrData, pcrSize);
                }
            }
        }
        saveIndex += stateSaved;
    }

    return;
}

//*** PCRIsStateSaved()
// This function indicates if the selected PCR is a PCR that is state saved
// on TPM2_Shutdown(STATE). The return value is based on PCR attributes.
//  Return Type: BOOL
//      TRUE(1)         PCR is state saved
//      FALSE(0)        PCR is not state saved
BOOL PCRIsStateSaved(TPMI_DH_PCR handle  // IN: PCR handle to be extended
)
{
    UINT32         pcr = handle - PCR_FIRST;
    PCR_Attributes currentPcrAttributes =
        _platPcr__GetPcrInitializationAttributes(pcr);

    if(currentPcrAttributes.stateSave == SET)
        return TRUE;
    else
        return FALSE;
}

//*** PCRIsResetAllowed()
// This function indicates if a PCR may be reset by the current command locality.
// The return value is based on PCR attributes, and not the PCR allocation.
//  Return Type: BOOL
//      TRUE(1)         TPM2_PCR_Reset is allowed
//      FALSE(0)        TPM2_PCR_Reset is not allowed
BOOL PCRIsResetAllowed(TPMI_DH_PCR handle  // IN: PCR handle to be extended
)
{
    UINT8          commandLocality;
    UINT8          localityBits = 1;
    UINT32         pcr          = handle - PCR_FIRST;
    PCR_Attributes currentPcrAttributes =
        _platPcr__GetPcrInitializationAttributes(pcr);

    // Check for the locality
    commandLocality = _plat__LocalityGet();

#ifdef DRTM_PCR
    // For a TPM that does DRTM, Reset is not allowed at locality 4
    if(commandLocality == 4)
```

```
        return FALSE;
#endif

    localityBits = localityBits << commandLocality;
    if((localityBits & currentPcrAttributes.resetLocality) == 0)
        return FALSE;
    else
        return TRUE;
}

//*** PCRChanged()
// This function checks a PCR handle to see if the attributes for the PCR are set
// so that any change to the PCR causes an increment of the pcrCounter. If it does,
// then the function increments the counter. Will also bump the counter if the
// handle is zero which means that PCR 0 can not be in the TCB group. Bump on zero
// is used by TPM2_Clear().
void PCRChanged(TPM_HANDLE pcrHandle  // IN: the handle of the PCR that changed.
)
{
    // For the reference implementation, the only change that does not cause
    // increment is a change to a PCR in the TCB group.
    if((pcrHandle == 0) || !PCRBelongsTCBGroup(pcrHandle))
    {
        gr.pcrCounter++;
        if(gr.pcrCounter == 0)
            FAIL(FATAL_ERROR_COUNTER_OVERFLOW);
    }
}

//*** PCRIsExtendAllowed()
// This function indicates a PCR may be extended at the current command locality.
// The return value is based on PCR attributes, and not the PCR allocation.
//  Return Type: BOOL
//      TRUE(1)         extend is allowed
//      FALSE(0)        extend is not allowed
BOOL PCRIsExtendAllowed(TPMI_DH_PCR handle  // IN: PCR handle to be extended
)
{
    UINT8          commandLocality;
    UINT8          localityBits = 1;
    UINT32         pcr          = handle - PCR_FIRST;
    PCR_Attributes currentPcrAttributes =
        _platPcr__GetPcrInitializationAttributes(pcr);

    // Check for the locality
    commandLocality = _plat__LocalityGet();
    localityBits    = localityBits << commandLocality;
    if((localityBits & currentPcrAttributes.extendLocality) == 0)
        return FALSE;
    else
        return TRUE;
}

//*** PCRExtend()
// This function is used to extend a PCR in a specific bank.
void PCRExtend(TPMI_DH_PCR   handle,  // IN: PCR handle to be extended
               TPMI_ALG_HASH hash,    // IN: hash algorithm of PCR
               UINT32        size,    // IN: size of data to be extended
               BYTE*         data     // IN: data to be extended
)
{
    BYTE*      pcrData;
    HASH_STATE hashState;
    UINT16     pcrSize;

    pcrData = GetPcrPointer(hash, handle - PCR_FIRST);
```

```
    // Extend PCR if it is allocated
    if(pcrData != NULL)
    {
        pcrSize = CryptHashGetDigestSize(hash);
        CryptHashStart(&hashState, hash);
        CryptDigestUpdate(&hashState, pcrSize, pcrData);
        CryptDigestUpdate(&hashState, size, data);
        CryptHashEnd(&hashState, pcrSize, pcrData);

        // PCR has changed so update the pcrCounter if necessary
        PCRChanged(handle);
    }

    return;
}

//*** PCRComputeCurrentDigest()
// This function computes the digest of the selected PCR.
//
// As a side-effect, 'selection' is modified so that only the implemented PCR
// will have their bits still set.
void PCRComputeCurrentDigest(
    TPMI_ALG_HASH       hashAlg,    // IN: hash algorithm to compute digest
    TPML_PCR_SELECTION* selection,  // IN/OUT: PCR selection (filtered on
                                    //     output)
    TPM2B_DIGEST* digest            // OUT: digest
)
{
    HASH_STATE          hashState;
    TPMS_PCR_SELECTION* select;
    BYTE*               pcrData;  // will point to a digest
    UINT32              pcrSize;
    UINT32              pcr;
    UINT32              i;

    // Initialize the hash
    digest->t.size = CryptHashStart(&hashState, hashAlg);
    pAssert(digest->t.size > 0 && digest->t.size < UINT16_MAX);

    // Iterate through the list of PCR selection structures
    for(i = 0; i < selection->count; i++)
    {
        // Point to the current selection
        select = &selection->pcrSelections[i];  // Point to the current selection
        FilterPcr(select);  // Clear out the bits for unimplemented PCR

        // Need the size of each digest
        pcrSize = CryptHashGetDigestSize(selection->pcrSelections[i].hash);

        // Iterate through the selection
        for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
        {
            if(IsPcrSelected(pcr, select))  // Is this PCR selected
            {
                // Get pointer to the digest data for the bank
                pcrData = GetPcrPointer(selection->pcrSelections[i].hash, pcr);
                pAssert(pcrData != NULL);
                CryptDigestUpdate(&hashState, pcrSize, pcrData);  // add to digest
            }
        }
    }
    // Complete hash stack
    CryptHashEnd2B(&hashState, &digest->b);

    return;
```

```
}

//*** PCRRead()
// This function is used to read a list of selected PCR.  If the requested PCR
// number exceeds the maximum number that can be output, the 'selection' is
// adjusted to reflect the actual output PCR.
void PCRRead(TPML_PCR_SELECTION* selection,  // IN/OUT: PCR selection (filtered on
                                             //     output)
             TPML_DIGEST* digest,            // OUT: digest
             UINT32*      pcrCounter  // OUT: the current value of PCR generation
                                      //     number
)
{
    TPMS_PCR_SELECTION* select;
    BYTE*               pcrData;  // will point to a digest
    UINT32              pcr;
    UINT32              i;

    digest->count = 0;

    // Iterate through the list of PCR selection structures
    for(i = 0; i < selection->count; i++)
    {
        // Point to the current selection
        select = &selection->pcrSelections[i];  // Point to the current selection
        FilterPcr(select);  // Clear out the bits for unimplemented PCR

        // Iterate through the selection
        for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
        {
            if(IsPcrSelected(pcr, select))  // Is this PCR selected
            {
                // Check if number of digest exceed upper bound
                if(digest->count > 7)
                {
                    // Clear rest of the current select bitmap
                    while(pcr < IMPLEMENTATION_PCR
                          // do not round up!
                          && (pcr / 8) < select->sizeofSelect)
                    {
                        // do not round up!
                        select->pcrSelect[pcr / 8] &= (BYTE) ~(1 << (pcr % 8));
                        pcr++;
                    }
                    // Exit inner loop
                    break;
                }
                // Need the size of each digest
                digest->digests[digest->count].t.size =
                    CryptHashGetDigestSize(selection->pcrSelections[i].hash);

                // Get pointer to the digest data for the bank
                pcrData = GetPcrPointer(selection->pcrSelections[i].hash, pcr);
                pAssert(pcrData != NULL);
                // Add to the data to digest
                MemoryCopy(digest->digests[digest->count].t.buffer,
                           pcrData,
                           digest->digests[digest->count].t.size);
                digest->count++;
            }
        }
        // If we exit inner loop because we have exceed the output upper bound
        if(digest->count > 7 && pcr < IMPLEMENTATION_PCR)
        {
            // Clear rest of the selection
            while(i < selection->count)
```

```
            {
                MemorySet(selection->pcrSelections[i].pcrSelect,
                          0,
                          selection->pcrSelections[i].sizeofSelect);
                i++;
            }
            // exit outer loop
            break;
        }
    }

    *pcrCounter = gr.pcrCounter;

    return;
}

//*** PCRAllocate()
// This function is used to change the PCR allocation.
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT        allocate failed
//      TPM_RC_PCR              improper allocation
TPM_RC
PCRAllocate(TPML_PCR_SELECTION* allocate,      // IN: required allocation
            UINT32*             maxPCR,        // OUT: Maximum number of PCR
            UINT32*             sizeNeeded,    // OUT: required space
            UINT32*             sizeAvailable  // OUT: available space
)
{
    UINT32             i, j, k;
    TPML_PCR_SELECTION newAllocate;
    // Initialize the flags to indicate if HCRTM PCR and DRTM PCR are allocated.
    BOOL pcrHcrtm = FALSE;
    BOOL pcrDrtm  = FALSE;

    // Create the expected new PCR allocation based on the existing allocation
    // and the new input:
    //  1. if a PCR bank does not appear in the new allocation, the existing
    //     allocation of this PCR bank will be preserved.
    //  2. if a PCR bank appears multiple times in the new allocation, only the
    //     last one will be in effect.
    newAllocate = gp.pcrAllocated;
    for(i = 0; i < allocate->count; i++)
    {
        for(j = 0; j < newAllocate.count; j++)
        {
            // If hash matches, the new allocation covers the old allocation
            // for this particular bank.
            // The assumption is the initial PCR allocation (from manufacture)
            // has all the supported hash algorithms with an assigned bank
            // (possibly empty).  So there must be a match for any new bank
            // allocation from the input.
            if(newAllocate.pcrSelections[j].hash == allocate->pcrSelections[i].hash)
            {
                newAllocate.pcrSelections[j] = allocate->pcrSelections[i];
                break;
            }
        }
        // The j loop must exit with a match.
        pAssert(j < newAllocate.count);
    }

    // Max PCR in a bank is MIN(implemented PCR, PCR with attributes defined)
    *maxPCR = _platPcr__NumberOfPcrs();
    if(*maxPCR > IMPLEMENTATION_PCR)
        *maxPCR = IMPLEMENTATION_PCR;
```

```
    // Compute required size for allocation
    *sizeNeeded = 0;
    for(i = 0; i < newAllocate.count; i++)
    {
        UINT32 digestSize = CryptHashGetDigestSize(newAllocate.pcrSelections[i].hash);
#if defined(DRTM_PCR)
        // Make sure that we end up with at least one DRTM PCR
        pcrDrtm = pcrDrtm
                  || TestBit(DRTM_PCR,
                             newAllocate.pcrSelections[i].pcrSelect,
                             newAllocate.pcrSelections[i].sizeofSelect);

#else  // if DRTM PCR is not required, indicate that the allocation is OK
        pcrDrtm = TRUE;
#endif

#if defined(HCRTM_PCR)
        // and one HCRTM PCR (since this is usually PCR 0...)
        pcrHcrtm = pcrHcrtm
                   || TestBit(HCRTM_PCR,
                              newAllocate.pcrSelections[i].pcrSelect,
                              newAllocate.pcrSelections[i].sizeofSelect);
#else
        pcrHcrtm = TRUE;
#endif
        for(j = 0; j < newAllocate.pcrSelections[i].sizeofSelect; j++)
        {
            BYTE mask = 1;
            for(k = 0; k < 8; k++)
            {
                if((newAllocate.pcrSelections[i].pcrSelect[j] & mask) != 0)
                    *sizeNeeded += digestSize;
                mask = mask << 1;
            }
        }
    }

    if(!pcrDrtm || !pcrHcrtm)
        return TPM_RC_PCR;

    // In this particular implementation, we always have enough space to
    // allocate PCR.  Different implementation may return a sizeAvailable less
    // than the sizeNeed.
    *sizeAvailable = sizeof(s_pcrs);

    // Save the required allocation to NV.  Note that after NV is written, the
    // PCR allocation in NV is no longer consistent with the RAM data
    // gp.pcrAllocated.  The NV version reflect the allocate after next
    // TPM_RESET, while the RAM version reflects the current allocation
    NV_WRITE_PERSISTENT(pcrAllocated, newAllocate);

    return TPM_RC_SUCCESS;
}

//*** PCRSetValue()
// This function is used to set the designated PCR in all banks to an initial value.
// The initial value is signed and will be sign extended into the entire PCR.
//
void PCRSetValue(TPM_HANDLE handle,       // IN: the handle of the PCR to set
                 INT8       initialValue  // IN: the value to set
)
{
    int           i;
    UINT32        pcr = handle - PCR_FIRST;
    TPMI_ALG_HASH hash;
    UINT16        digestSize;
```

```
    BYTE*         pcrData;

    // Iterate supported PCR bank algorithms to reset
    for(i = 0; i < HASH_COUNT; i++)
    {
        hash = CryptHashGetAlgByIndex(i);
        // Prevent runaway
        if(hash == TPM_ALG_NULL)
            break;

        // Get a pointer to the data
        pcrData = GetPcrPointer(gp.pcrAllocated.pcrSelections[i].hash, pcr);

        // If the PCR is allocated
        if(pcrData != NULL)
        {
            // And the size of the digest
            digestSize = CryptHashGetDigestSize(hash);

            // Set the LSO to the input value
            pcrData[digestSize - 1] = initialValue;

            // Sign extend
            if(initialValue >= 0)
                MemorySet(pcrData, 0, digestSize - 1);
            else
                MemorySet(pcrData, -1, digestSize - 1);
        }
    }
}

//*** PCRResetDynamics
// This function is used to reset a dynamic PCR to 0.  This function is used in
// DRTM sequence.
void PCRResetDynamics(void)
{
    UINT32 pcr, i;

    // Initialize PCR values
    for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        // Iterate each hash algorithm bank
        for(i = 0; i < gp.pcrAllocated.count; i++)
        {
            BYTE*          pcrData;
            UINT32         pcrSize;
            PCR_Attributes currentPcrAttributes =
                _platPcr__GetPcrInitializationAttributes(pcr);

            pcrData = GetPcrPointer(gp.pcrAllocated.pcrSelections[i].hash, pcr);

            if(pcrData != NULL)
            {
                pcrSize =
                    CryptHashGetDigestSize(gp.pcrAllocated.pcrSelections[i].hash);

                // Reset PCR
                // Any PCR can be reset by locality 4 should be reset to 0
                if((currentPcrAttributes.resetLocality & 0x10) != 0)
                    MemorySet(pcrData, 0, pcrSize);
            }
        }
    }
    return;
}
```

```
//*** PCRCapGetAllocation()
// This function is used to get the current allocation of PCR banks.
//  Return Type: TPMI_YES_NO
//      YES         if the return count is 0
//      NO          if the return count is not 0
TPMI_YES_NO
PCRCapGetAllocation(UINT32              count,        // IN: count of return
                    TPML_PCR_SELECTION* pcrSelection  // OUT: PCR allocation list
)
{
    if(count == 0)
    {
        pcrSelection->count = 0;
        return YES;
    }
    else
    {
        *pcrSelection = gp.pcrAllocated;
        return NO;
    }
}

//*** PCRSetSelectBit()
// This function sets a bit in a bitmap array.
static void PCRSetSelectBit(UINT32 pcr,    // IN: PCR number
                            BYTE*  bitmap  // OUT: bit map to be set
)
{
    bitmap[pcr / 8] |= (1 << (pcr % 8));
    return;
}

//*** PCRGetProperty()
// This function returns the selected PCR property.
//  Return Type: BOOL
//      TRUE(1)         the property type is implemented
//      FALSE(0)        the property type is not implemented
BOOL PCRGetProperty(TPM_PT_PCR property, TPMS_TAGGED_PCR_SELECT* select)
{
    UINT32 pcr;
    UINT32 groupIndex;

    select->tag = property;
    // Always set the bitmap to be the size of all PCR
    select->sizeofSelect = (IMPLEMENTATION_PCR + 7) / 8;

    // Initialize bitmap
    MemorySet(select->pcrSelect, 0, select->sizeofSelect);

    // Collecting properties
    for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        PCR_Attributes currentPcrAttributes =
            _platPcr__GetPcrInitializationAttributes(pcr);

        switch(property)
        {
            case TPM_PT_PCR_SAVE:
                if(currentPcrAttributes.stateSave == SET)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L0:
                if((currentPcrAttributes.extendLocality & 0x01) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L0:
```

```
                if((currentPcrAttributes.resetLocality & 0x01) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L1:
                if((currentPcrAttributes.extendLocality & 0x02) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L1:
                if((currentPcrAttributes.resetLocality & 0x02) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L2:
                if((currentPcrAttributes.extendLocality & 0x04) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L2:
                if((currentPcrAttributes.resetLocality & 0x04) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L3:
                if((currentPcrAttributes.extendLocality & 0x08) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L3:
                if((currentPcrAttributes.resetLocality & 0x08) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L4:
                if((currentPcrAttributes.extendLocality & 0x10) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L4:
                if((currentPcrAttributes.resetLocality & 0x10) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_DRTM_RESET:
                // DRTM reset PCRs are the PCR reset by locality 4
                if((currentPcrAttributes.resetLocality & 0x10) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
#if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
            case TPM_PT_PCR_POLICY:
                if(PCRBelongsPolicyGroup(pcr + PCR_FIRST, &groupIndex))
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
#endif
#if defined NUM_AUTHVALUE_PCR_GROUP && NUM_AUTHVALUE_PCR_GROUP > 0
            case TPM_PT_PCR_AUTH:
                if(PCRBelongsAuthGroup(pcr + PCR_FIRST, &groupIndex))
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
#endif
#if ENABLE_PCR_NO_INCREMENT == YES
            case TPM_PT_PCR_NO_INCREMENT:
                if(PCRBelongsTCBGroup(pcr + PCR_FIRST))
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
#endif
            default:
                // If property is not supported, stop scanning PCR attributes
                // and return.
                return FALSE;
                break;
        }
    }
    return TRUE;
```

```
}

//*** PCRCapGetProperties()
// This function returns a list of PCR properties starting at 'property'.
//  Return Type: TPMI_YES_NO
//      YES         if no more property is available
//      NO          if there are more properties not reported
TPMI_YES_NO
PCRCapGetProperties(TPM_PT_PCR property,  // IN: the starting PCR property
                    UINT32     count,     // IN: count of returned properties
                    TPML_TAGGED_PCR_PROPERTY* select  // OUT: PCR select
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;

    // Initialize output property list
    select->count = 0;

    // The maximum count of properties we may return is MAX_PCR_PROPERTIES
    if(count > MAX_PCR_PROPERTIES)
        count = MAX_PCR_PROPERTIES;

    // TPM_PT_PCR_FIRST is defined as 0 in spec.  It ensures that property
    // value would never be less than TPM_PT_PCR_FIRST
    MUST_BE(TPM_PT_PCR_FIRST == 0);

    // Iterate PCR properties. TPM_PT_PCR_LAST is the index of the last property
    // implemented on the TPM.
    for(i = property; i <= TPM_PT_PCR_LAST; i++)
    {
        if(select->count < count)
        {
            // If we have not filled up the return list, add more properties to it
            if(PCRGetProperty(i, &select->pcrProperty[select->count]))
                // only increment if the property is implemented
                select->count++;
        }
        else
        {
            // If the return list is full but we still have properties
            // available, report this and stop iterating.
            more = YES;
            break;
        }
    }
    return more;
}

//*** PCRCapGetHandles()
// This function is used to get a list of handles of PCR, started from 'handle'.
// If 'handle' exceeds the maximum PCR handle range, an empty list will be
// returned and the return value will be NO.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
PCRCapGetHandles(TPMI_DH_PCR  handle,     // IN: start handle
                 UINT32       count,      // IN: count of returned handles
                 TPML_HANDLE* handleList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;

    pAssert(HandleGetType(handle) == TPM_HT_PCR);
```

```
    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES)
        count = MAX_CAP_HANDLES;

    // Iterate PCR handle range
    for(i = handle & HR_HANDLE_MASK; i <= PCR_LAST; i++)
    {
        if(handleList->count < count)
        {
            // If we have not filled up the return list, add this PCR
            // handle to it
            handleList->handle[handleList->count] = i + PCR_FIRST;
            handleList->count++;
        }
        else
        {
            // If the return list is full but we still have PCR handle
            // available, report this and stop iterating
            more = YES;
            break;
        }
    }
    return more;
}

//*** PCRCapGetOneHandle()
// This function is used to check whether a PCR handle exists.
BOOL PCRCapGetOneHandle(TPMI_DH_PCR handle)  // IN: handle
{
    pAssert(HandleGetType(handle) == TPM_HT_PCR);

    if((handle & HR_HANDLE_MASK) <= PCR_LAST)
    {
        return TRUE;
    }
    return FALSE;
}
```

7.177 /tpm/src/subsystem/PP.c

```
//** Introduction
// This file contains the functions that support the physical presence operations
// of the TPM.

//** Includes

#include "Tpm.h"

//** Functions

//*** PhysicalPresencePreInstall_Init()
// This function is used to initialize the array of commands that always require
// confirmation with physical presence. The array is an array of bits that
// has a correspondence with the command code.
//
// This command should only ever be executable in a manufacturing setting or in
// a simulation.
//
// When set, these cannot be cleared.
//
void PhysicalPresencePreInstall_Init(void)
```

```
{
    COMMAND_INDEX commandIndex;
    // Clear all the PP commands
    MemorySet(&gp.ppList, 0, sizeof(gp.ppList));

    // Any command that is PP_REQUIRED should be SET
    for(commandIndex = 0; commandIndex < COMMAND_COUNT; commandIndex++)
    {
        if(s_commandAttributes[commandIndex] & IS_IMPLEMENTED
           && s_commandAttributes[commandIndex] & PP_REQUIRED)
            SET_BIT(commandIndex, gp.ppList);
    }
    // Write PP list to NV
    NV_SYNC_PERSISTENT(ppList);
    return;
}

//*** PhysicalPresenceCommandSet()
// This function is used to set the indicator that a command requires
// PP confirmation.
void PhysicalPresenceCommandSet(TPM_CC commandCode  // IN: command code
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);

    // if the command isn't implemented, the do nothing
    if(commandIndex == UNIMPLEMENTED_COMMAND_INDEX)
        return;

    // only set the bit if this is a command for which PP is allowed
    if(s_commandAttributes[commandIndex] & PP_COMMAND)
        SET_BIT(commandIndex, gp.ppList);
    return;
}

//*** PhysicalPresenceCommandClear()
// This function is used to clear the indicator that a command requires PP
// confirmation.
void PhysicalPresenceCommandClear(TPM_CC commandCode  // IN: command code
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);

    // If the command isn't implemented, then don't do anything
    if(commandIndex == UNIMPLEMENTED_COMMAND_INDEX)
        return;

    // Only clear the bit if the command does not require PP
    if((s_commandAttributes[commandIndex] & PP_REQUIRED) == 0)
        CLEAR_BIT(commandIndex, gp.ppList);

    return;
}

//*** PhysicalPresenceIsRequired()
// This function indicates if PP confirmation is required for a command.
//  Return Type: BOOL
//      TRUE(1)         physical presence is required
//      FALSE(0)        physical presence is not required
BOOL PhysicalPresenceIsRequired(COMMAND_INDEX commandIndex  // IN: command index
)
{
    // Check the bit map.  If the bit is SET, PP authorization is required
    return (TEST_BIT(commandIndex, gp.ppList));
}
```

```
//*** PhysicalPresenceCapGetCCList()
// This function returns a list of commands that require PP confirmation. The
// list starts from the first implemented command that has a command code that
// the same or greater than 'commandCode'.
//  Return Type: TPMI_YES_NO
//      YES         if there are more command codes available
//      NO          all the available command codes have been returned
TPMI_YES_NO
PhysicalPresenceCapGetCCList(TPM_CC   commandCode,  // IN: start command code
                             UINT32   count,        // IN: count of returned TPM_CC
                             TPML_CC* commandList   // OUT: list of TPM_CC
)
{
    TPMI_YES_NO   more = NO;
    COMMAND_INDEX commandIndex;

    // Initialize output handle list
    commandList->count = 0;

    // The maximum count of command we may return is MAX_CAP_CC
    if(count > MAX_CAP_CC)
        count = MAX_CAP_CC;

    // Collect PP commands
    for(commandIndex = GetClosestCommandIndex(commandCode);
        commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
        commandIndex = GetNextCommandIndex(commandIndex))
    {
        if(PhysicalPresenceIsRequired(commandIndex))
        {
            if(commandList->count < count)
            {
                // If we have not filled up the return list, add this command
                // code to it
                commandList->commandCodes[commandList->count] =
                    GetCommandCode(commandIndex);
                commandList->count++;
            }
            else
            {
                // If the return list is full but we still have PP command
                // available, report this and stop iterating
                more = YES;
                break;
            }
        }
    }
    return more;
}

//*** PhysicalPresenceCapGetOneCC()
// This function returns true if the command requires Physical Presence.
BOOL PhysicalPresenceCapGetOneCC(TPM_CC commandCode)  // IN: command code
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);
    if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
    {
        return PhysicalPresenceIsRequired(commandIndex);
    }
    return FALSE;
}
```

7.178 /tpm/src/subsystem/Session.c

```
//**Introduction
```

```
/*
    The code in this file is used to manage the session context counter.
    The scheme implemented here is a "truncated counter".
    This scheme allows the TPM to not need TPM_SU_CLEAR for a
    very long period of time and still not have the context
    count for a session repeated.

    The counter (contextCounter)in this implementation is a UINT64 but
    can be smaller.  The "tracking array" (contextArray) only
    has 16-bits per context.  The tracking array is the data
    that needs to be saved and restored across TPM_SU_STATE so that
    sessions are not lost when the system enters the sleep state.
    Also, when the TPM is active, the tracking array is kept in
    RAM making it important that the number of bytes for each
    entry be kept as small as possible.

    The TPM prevents "collisions" of these truncated values by
    not allowing a contextID to be assigned if it would be the
    same as an existing value.  Since the array holds 16 bits,
    after a context has been saved, an additional 2^16-1 contexts
    may be saved before the count would again match.  The normal
    expectation is that the context will be flushed before its count
    value is needed again but it is always possible to have long-lived
    sessions.

    The contextID is assigned when the context is saved (TPM2_ContextSave()).
    At that time, the TPM will compare the low-order 16 bits of
    contextCounter to the existing values in contextArray and if one
    matches, the TPM will return TPM_RC_CONTEXT_GAP (by construction,
    the entry that contains the matching value is the oldest
    context).

    The expected remediation by the TRM is to load the oldest saved
    session context (the one found by the TPM), and save it.  Since loading
    the oldest session also eliminates its contextID value from
    contextArray, there TPM will always be able to load and save the oldest
    existing context.

    In the worst case, software may have to load and save several contexts
    in order to save an additional one.  This should happen very infrequently.

    When the TPM searches contextArray and finds that none of the contextIDs
    match the low-order 16-bits of contextCount, the TPM can copy the low bits
    to the contextArray associated with the session, and increment contextCount.

    There is one entry in contextArray for each of the active sessions
    allowed by the TPM implementation.  This array contains either a
    context count, an index, or a value indicating the slot is available (0).

    The index into the contextArray is the handle for the session with the region
    selector byte of the session set to zero.  If an entry in contextArray contains
    0, then the corresponding handle may be assigned to a session.  If the entry
    contains a value that is less than or equal to the number of loaded sessions
    for the TPM, then the array entry is the slot in which the context is loaded.

    EXAMPLE:    If the TPM allows 8 loaded sessions, then the slot numbers would
    be 1-8 and a contextArrary value in that range would represent the loaded
    session.

    NOTE:   When the TPM firmware determines that the array entry is for a loaded
    session, it will subtract 1 to create the zero-based slot number.

    There is one significant corner case in this scheme.  When the contextCount
    is equal to a value in the contextArray, the oldest session needs to be
    recycled or flushed. In order to recycle the session, it must be loaded.
    To be loaded, there must be an available slot.  Rather than require that a
```

```
    spare slot be available all the time, the TPM will check to see if the
    contextCount is equal to some value in the contextArray when a session is
    created.  This prevents the last session slot from being used when it
    is likely that a session will need to be recycled.

    If a TPM with both 1.2 and 2.0 functionality uses this scheme for both
    1.2 and 2.0 sessions, and the list of active contexts is read with
    TPM_GetCapabiltiy(), the TPM will create 32-bit representations of the
    list that contains 16-bit values (the TPM2_GetCapability() returns a list
    of handles for active sessions rather than a list of contextID).  The full
    contextID has high-order bits that are either the same as the current
    contextCount or one less.  It is one less if the 16-bits
    of the contextArray has a value that is larger than the low-order 16 bits
    of contextCount.
*/

//** Includes, Defines, and Local Variables
#define SESSION_C
#include "Tpm.h"

//** File Scope Function -- ContextIdSetOldest()
/*
    This function is called when the oldest contextID is being loaded or deleted.
    Once a saved context becomes the oldest, it stays the oldest until it is
    deleted.

    Finding the oldest is a bit tricky.  It is not just the numeric comparison of
    values but is dependent on the value of contextCounter.

    Assume we have a small contextArray with 8, 4-bit values with values 1 and 2
    used to indicate the loaded context slot number.  Also assume that the array
    contains hex values of (0 0 1 0 3 0 9 F) and that the contextCounter is an
    8-bit counter with a value of 0x37. Since the low nibble is 7, that means
    that values above 7 are older than values below it and, in this example,
    9 is the oldest value.

    Note if we subtract the counter value, from each slot that contains a saved
    contextID we get (- - - - B - 2 - 8) and the oldest entry is now easy to find.
*/
static void ContextIdSetOldest(void)
{
    CONTEXT_SLOT lowBits;
    CONTEXT_SLOT entry;
    CONTEXT_SLOT smallest = ((CONTEXT_SLOT)~0);
    UINT32       i;

    // Set oldestSaveContext to a value indicating none assigned
    s_oldestSavedSession = MAX_ACTIVE_SESSIONS + 1;

    lowBits              = (CONTEXT_SLOT)gr.contextCounter;
    for(i = 0; i < MAX_ACTIVE_SESSIONS; i++)
    {
        entry = gr.contextArray[i];

        // only look at entries that are saved contexts
        if(entry > MAX_LOADED_SESSIONS)
        {
            // Use a less than or equal in case the oldest
            // is brand new (= lowBits-1) and equal to our initial
            // value for smallest.
            if(((CONTEXT_SLOT)(entry - lowBits)) <= smallest)
            {
                smallest             = (entry - lowBits);
                s_oldestSavedSession = i;
            }
        }
```

```
    }
    // When we finish, either the s_oldestSavedSession still has its initial
    // value, or it has the index of the oldest saved context.
}

//** Startup Function -- SessionStartup()
// This function initializes the session subsystem on TPM2_Startup().
BOOL SessionStartup(STARTUP_TYPE type)
{
    UINT32 i;

    // Initialize session slots.  At startup, all the in-memory session slots
    // are cleared and marked as not occupied
    for(i = 0; i < MAX_LOADED_SESSIONS; i++)
        s_sessions[i].occupied = FALSE;  // session slot is not occupied

    // The free session slots the number of maximum allowed loaded sessions
    s_freeSessionSlots = MAX_LOADED_SESSIONS;

    // Initialize context ID data.  On a ST_SAVE or hibernate sequence, it will
    // scan the saved array of session context counts, and clear any entry that
    // references a session that was in memory during the state save since that
    // memory was not preserved over the ST_SAVE.
    if(type == SU_RESUME || type == SU_RESTART)
    {
        // On ST_SAVE we preserve the contexts that were saved but not the ones
        // in memory
        for(i = 0; i < MAX_ACTIVE_SESSIONS; i++)
        {
            // If the array value is unused or references a loaded session then
            // that loaded session context is lost and the array entry is
            // reclaimed.
            if(gr.contextArray[i] <= MAX_LOADED_SESSIONS)
                gr.contextArray[i] = 0;
        }
        // Find the oldest session in context ID data and set it in
        // s_oldestSavedSession
        ContextIdSetOldest();
    }
    else
    {
        // For STARTUP_CLEAR, clear out the contextArray
        for(i = 0; i < MAX_ACTIVE_SESSIONS; i++)
            gr.contextArray[i] = 0;

        // reset the context counter
        gr.contextCounter = MAX_LOADED_SESSIONS + 1;

        // Initialize oldest saved session
        s_oldestSavedSession = MAX_ACTIVE_SESSIONS + 1;
    }
    return TRUE;
}

//**********************************************
//** Access Functions
//**********************************************

//*** SessionIsLoaded()
// This function test a session handle references a loaded session.  The handle
// must have previously been checked to make sure that it is a valid handle for
// an authorization session.
// NOTE:    A PWAP authorization does not have a session.
//
//  Return Type: BOOL
//      TRUE(1)         session is loaded
```

```
//      FALSE(0)        session is not loaded
//
BOOL SessionIsLoaded(TPM_HANDLE handle  // IN: session handle
)
{
    pAssert(HandleGetType(handle) == TPM_HT_POLICY_SESSION
            || HandleGetType(handle) == TPM_HT_HMAC_SESSION);

    handle = handle & HR_HANDLE_MASK;

    // if out of range of possible active session, or not assigned to a loaded
    // session return false
    if(handle >= MAX_ACTIVE_SESSIONS || gr.contextArray[handle] == 0
       || gr.contextArray[handle] > MAX_LOADED_SESSIONS)
        return FALSE;

    return TRUE;
}

//*** SessionIsSaved()
// This function test a session handle references a saved session.  The handle
// must have previously been checked to make sure that it is a valid handle for
// an authorization session.
// NOTE:    An password authorization does not have a session.
//
// This function requires that the handle be a valid session handle.
//
//  Return Type: BOOL
//      TRUE(1)         session is saved
//      FALSE(0)        session is not saved
//
BOOL SessionIsSaved(TPM_HANDLE handle  // IN: session handle
)
{
    pAssert(HandleGetType(handle) == TPM_HT_POLICY_SESSION
            || HandleGetType(handle) == TPM_HT_HMAC_SESSION);

    handle = handle & HR_HANDLE_MASK;
    // if out of range of possible active session, or not assigned, or
    // assigned to a loaded session, return false
    if(handle >= MAX_ACTIVE_SESSIONS || gr.contextArray[handle] == 0
       || gr.contextArray[handle] <= MAX_LOADED_SESSIONS)
        return FALSE;

    return TRUE;
}

//*** SequenceNumberForSavedContextIsValid()
// This function validates that the sequence number and handle value within a
// saved context are valid.
BOOL SequenceNumberForSavedContextIsValid(
    TPMS_CONTEXT* context  // IN: pointer to a context structure to be
                           //     validated
)
{
#define MAX_CONTEXT_GAP ((UINT64)((CONTEXT_SLOT)~0) + 1)

    TPM_HANDLE handle = context->savedHandle & HR_HANDLE_MASK;

    if(  // Handle must be with the range of active sessions
        handle >= MAX_ACTIVE_SESSIONS
        // the array entry must be for a saved context
        || gr.contextArray[handle] <= MAX_LOADED_SESSIONS
        // the array entry must agree with the sequence number
        || gr.contextArray[handle] != (CONTEXT_SLOT)context->sequence
        // the provided sequence number has to be less than the current counter
```

```
        || context->sequence > gr.contextCounter
        // but not so much that it could not be a valid sequence number
        || gr.contextCounter - context->sequence > MAX_CONTEXT_GAP)
        return FALSE;

    return TRUE;
}

//*** SessionPCRValueIsCurrent()
//
// This function is used to check if PCR values have been updated since the
// last time they were checked in a policy session.
//
// This function requires the session is loaded.
//  Return Type: BOOL
//      TRUE(1)         PCR value is current
//      FALSE(0)        PCR value is not current
BOOL SessionPCRValueIsCurrent(SESSION* session  // IN: session structure
)
{
    if(session->pcrCounter != 0 && session->pcrCounter != gr.pcrCounter)
        return FALSE;
    else
        return TRUE;
}

//*** SessionGet()
// This function returns a pointer to the session object associated with a
// session handle.
//
// The function requires that the session is loaded.
SESSION* SessionGet(TPM_HANDLE handle  // IN: session handle
)
{
    size_t       slotIndex;
    CONTEXT_SLOT sessionIndex;

    pAssert(HandleGetType(handle) == TPM_HT_POLICY_SESSION
            || HandleGetType(handle) == TPM_HT_HMAC_SESSION);

    slotIndex = handle & HR_HANDLE_MASK;

    pAssert(slotIndex < MAX_ACTIVE_SESSIONS);

    // get the contents of the session array.  Because session is loaded, we
    // should always get a valid sessionIndex
    sessionIndex = gr.contextArray[slotIndex] - 1;

    pAssert(sessionIndex < MAX_LOADED_SESSIONS);

    return &s_sessions[sessionIndex].session;
}

//**********************************************
//** Utility Functions
//**********************************************

//*** ContextIdSessionCreate()
//
//  This function is called when a session is created.  It will check
//  to see if the current gap would prevent a context from being saved.  If
//  so it will return TPM_RC_CONTEXT_GAP.  Otherwise, it will try to find
//  an open slot in contextArray, set contextArray to the slot.
//
//  This routine requires that the caller has determined the session array
//  index for the session.
```

```
//
//  Return Type: TPM_RC
//      TPM_RC_CONTEXT_GAP      can't assign a new contextID until the oldest
//                              saved session context is recycled
//      TPM_RC_SESSION_HANDLE   there is no slot available in the context array
//                              for tracking of this session context
static TPM_RC ContextIdSessionCreate(
    TPM_HANDLE* handle,  // OUT: receives the assigned handle. This will
                         //     be an index that must be adjusted by the
                         //     caller according to the type of the
                         //     session created
    UINT32 sessionIndex  // IN: The session context array entry that will
                         //     be occupied by the created session
)
{
    pAssert(sessionIndex < MAX_LOADED_SESSIONS);

    // check to see if creating the context is safe
    // Is this going to be an assignment for the last session context
    // array entry?  If so, then there will be no room to recycle the
    // oldest context if needed.  If the gap is not at maximum, then
    // it will be possible to save a context if it becomes necessary.
    if(s_oldestSavedSession < MAX_ACTIVE_SESSIONS && s_freeSessionSlots == 1)
    {
        // See if the gap is at maximum
        // The current value of the contextCounter will be assigned to the next
        // saved context. If the value to be assigned would make the same as an
        // existing context, then we can't use it because of the ambiguity it would
        // create.
        if((CONTEXT_SLOT)gr.contextCounter == gr.contextArray[s_oldestSavedSession])
            return TPM_RC_CONTEXT_GAP;
    }

    // Find an unoccupied entry in the contextArray
    for(*handle = 0; *handle < MAX_ACTIVE_SESSIONS; (*handle)++)
    {
        if(gr.contextArray[*handle] == 0)
        {
            // indicate that the session associated with this handle
            // references a loaded session
            gr.contextArray[*handle] = (CONTEXT_SLOT)(sessionIndex + 1);
            return TPM_RC_SUCCESS;
        }
    }
    return TPM_RC_SESSION_HANDLES;
}

//*** SessionCreate()
//
//  This function does the detailed work for starting an authorization session.
//  This is done in a support routine rather than in the action code because
//  the session management may differ in implementations.  This implementation
//  uses a fixed memory allocation to hold sessions and a fixed allocation
//  to hold the contextID for the saved contexts.
//
//  Return Type: TPM_RC
//      TPM_RC_CONTEXT_GAP          need to recycle sessions
//      TPM_RC_SESSION_HANDLE       active session space is full
//      TPM_RC_SESSION_MEMORY       loaded session space is full
TPM_RC
SessionCreate(TPM_SE         sessionType,    // IN: the session type
              TPMI_ALG_HASH  authHash,       // IN: the hash algorithm
              TPM2B_NONCE*   nonceCaller,    // IN: initial nonceCaller
              TPMT_SYM_DEF*  symmetric,      // IN: the symmetric algorithm
              TPMI_DH_ENTITY bind,           // IN: the bind object
              TPM2B_DATA*    seed,           // IN: seed data
```

```
                TPM_HANDLE*     sessionHandle,  // OUT: the session handle
                TPM2B_NONCE*    nonceTpm        // OUT: the session nonce
)
{
    TPM_RC       result = TPM_RC_SUCCESS;
    CONTEXT_SLOT slotIndex;
    SESSION*     session = NULL;

    pAssert(sessionType == TPM_SE_HMAC || sessionType == TPM_SE_POLICY
            || sessionType == TPM_SE_TRIAL);

    // If there are no open spots in the session array, then no point in searching
    if(s_freeSessionSlots == 0)
        return TPM_RC_SESSION_MEMORY;

    // Find a space for loading a session
    for(slotIndex = 0; slotIndex < MAX_LOADED_SESSIONS; slotIndex++)
    {
        // Is this available?
        if(s_sessions[slotIndex].occupied == FALSE)
        {
            session = &s_sessions[slotIndex].session;
            break;
        }
    }
    // if no spot found, then this is an internal error
    if(slotIndex >= MAX_LOADED_SESSIONS)
        FAIL(FATAL_ERROR_INTERNAL);

    // Call context ID function to get a handle.  TPM_RC_SESSION_HANDLE may be
    // returned from ContextIdHandelAssign()
    result = ContextIdSessionCreate(sessionHandle, slotIndex);
    if(result != TPM_RC_SUCCESS)
        return result;

    //*** Only return from this point on is TPM_RC_SUCCESS

    // Can now indicate that the session array entry is occupied.
    s_freeSessionSlots--;
    s_sessions[slotIndex].occupied = TRUE;

    // Initialize the session data
    MemorySet(session, 0, sizeof(SESSION));

    // Initialize internal session data
    session->authHashAlg = authHash;
    // Initialize session type
    if(sessionType == TPM_SE_HMAC)
    {
        *sessionHandle += HMAC_SESSION_FIRST;
    }
    else
    {
        *sessionHandle += POLICY_SESSION_FIRST;

        // For TPM_SE_POLICY or TPM_SE_TRIAL
        session->attributes.isPolicy = SET;
        if(sessionType == TPM_SE_TRIAL)
            session->attributes.isTrialPolicy = SET;

        SessionSetStartTime(session);

        // Initialize policyDigest.  policyDigest is initialized with a string of 0
        // of session algorithm digest size. Since the session is already clear.
        // Just need to set the size
        session->u2.policyDigest.t.size =
```

```
            CryptHashGetDigestSize(session->authHashAlg);
    }
    // Create initial session nonce
    session->nonceTPM.t.size = nonceCaller->t.size;
    CryptRandomGenerate(session->nonceTPM.t.size, session->nonceTPM.t.buffer);
    MemoryCopy2B(&nonceTpm->b, &session->nonceTPM.b, sizeof(nonceTpm->t.buffer));

    // Set up session parameter encryption algorithm
    session->symmetric = *symmetric;

    // If there is a bind object or a session secret, then need to compute
    // a sessionKey.
    if(bind != TPM_RH_NULL || seed->t.size != 0)
    {
        // sessionKey = KDFa(hash, (authValue || seed), "ATH", nonceTPM,
        //                   nonceCaller, bits)
        // The HMAC key for generating the sessionSecret can be the concatenation
        // of an authorization value and a seed value
        TPM2B_TYPE(KEY, (sizeof(TPMT_HA) + sizeof(seed->t.buffer)));
        TPM2B_KEY key;

        // Get hash size, which is also the length of sessionKey
        session->sessionKey.t.size = CryptHashGetDigestSize(session->authHashAlg);

        // Get authValue of associated entity
        EntityGetAuthValue(bind, (TPM2B_AUTH*)&key);
        pAssert(key.t.size + seed->t.size <= sizeof(key.t.buffer));

        // Concatenate authValue and seed
        MemoryConcat2B(&key.b, &seed->b, sizeof(key.t.buffer));

        // Compute the session key
        CryptKDFa(session->authHashAlg,
                  &key.b,
                  SESSION_KEY,
                  &session->nonceTPM.b,
                  &nonceCaller->b,
                  session->sessionKey.t.size * 8,
                  session->sessionKey.t.buffer,
                  NULL,
                  FALSE);
    }

    // Copy the name of the entity that the HMAC session is bound to
    // Policy session is not bound to an entity
    if(bind != TPM_RH_NULL && sessionType == TPM_SE_HMAC)
    {
        session->attributes.isBound = SET;
        SessionComputeBoundEntity(bind, &session->u1.boundEntity);
    }
    // If there is a bind object and it is subject to DA, then use of this session
    // is subject to DA regardless of how it is used.
    session->attributes.isDaBound = (bind != TPM_RH_NULL)
                                    && (IsDAExempted(bind) == FALSE);

    // If the session is bound, then check to see if it is bound to lockoutAuth
    session->attributes.isLockoutBound = (session->attributes.isDaBound == SET)
                                         && (bind == TPM_RH_LOCKOUT);
    return TPM_RC_SUCCESS;
}

//*** SessionContextSave()
// This function is called when a session context is to be saved.  The
// contextID of the saved session is returned.  If no contextID can be
// assigned, then the routine returns TPM_RC_CONTEXT_GAP.
// If the function completes normally, the session slot will be freed.
```

```
//
// This function requires that 'handle' references a loaded session.
// Otherwise, it should not be called at the first place.
//
//  Return Type: TPM_RC
//      TPM_RC_CONTEXT_GAP              a contextID could not be assigned
//      TPM_RC_TOO_MANY_CONTEXTS        the counter maxed out
//
TPM_RC
SessionContextSave(TPM_HANDLE       handle,    // IN: session handle
                   CONTEXT_COUNTER* contextID  // OUT: assigned contextID
)
{
    UINT32       contextIndex;
    CONTEXT_SLOT slotIndex;

    pAssert(SessionIsLoaded(handle));

    // check to see if the gap is already maxed out
    // Need to have a saved session
    if(s_oldestSavedSession < MAX_ACTIVE_SESSIONS
       // if the oldest saved session has the same value as the low bits
       // of the contextCounter, then the GAP is maxed out.
       && gr.contextArray[s_oldestSavedSession] == (CONTEXT_SLOT)gr.contextCounter)
        return TPM_RC_CONTEXT_GAP;

    // if the caller wants the context counter, set it
    if(contextID != NULL)
        *contextID = gr.contextCounter;

    contextIndex = handle & HR_HANDLE_MASK;
    pAssert(contextIndex < MAX_ACTIVE_SESSIONS);

    // Extract the session slot number referenced by the contextArray
    // because we are going to overwrite this with the low order
    // contextID value.
    slotIndex = gr.contextArray[contextIndex] - 1;

    // Set the contextID for the contextArray
    gr.contextArray[contextIndex] = (CONTEXT_SLOT)gr.contextCounter;

    // Increment the counter
    gr.contextCounter++;

    // In the unlikely event that the 64-bit context counter rolls over...
    if(gr.contextCounter == 0)
    {
        // back it up
        gr.contextCounter--;
        // return an error
        return TPM_RC_TOO_MANY_CONTEXTS;
    }
    // if the low-order bits wrapped, need to advance the value to skip over
    // the values used to indicate that a session is loaded
    if(((CONTEXT_SLOT)gr.contextCounter) == 0)
        gr.contextCounter += MAX_LOADED_SESSIONS + 1;

    // If no other sessions are saved, this is now the oldest.
    if(s_oldestSavedSession >= MAX_ACTIVE_SESSIONS)
        s_oldestSavedSession = contextIndex;

    // Mark the session slot as unoccupied
    s_sessions[slotIndex].occupied = FALSE;

    // and indicate that there is an additional open slot
    s_freeSessionSlots++;
```

```
    return TPM_RC_SUCCESS;
}

//*** SessionContextLoad()
// This function is used to load a session from saved context.  The session
// handle must be for a saved context.
//
// If the gap is at a maximum, then the only session that can be loaded is
// the oldest session, otherwise TPM_RC_CONTEXT_GAP is returned.
//
// This function requires that 'handle' references a valid saved session.
//
//  Return Type: TPM_RC
//      TPM_RC_SESSION_MEMORY       no free session slots
//      TPM_RC_CONTEXT_GAP          the gap count is maximum and this
//                                  is not the oldest saved context
//
TPM_RC
SessionContextLoad(SESSION_BUF* session,  // IN: session structure from saved context
                   TPM_HANDLE*  handle    // IN/OUT: session handle
)
{
    UINT32       contextIndex;
    CONTEXT_SLOT slotIndex;

    pAssert(HandleGetType(*handle) == TPM_HT_POLICY_SESSION
            || HandleGetType(*handle) == TPM_HT_HMAC_SESSION);

    // Don't bother looking if no openings
    if(s_freeSessionSlots == 0)
        return TPM_RC_SESSION_MEMORY;

    // Find a free session slot to load the session
    for(slotIndex = 0; slotIndex < MAX_LOADED_SESSIONS; slotIndex++)
        if(s_sessions[slotIndex].occupied == FALSE)
            break;

    // if no spot found, then this is an internal error
    pAssert(slotIndex < MAX_LOADED_SESSIONS);

    contextIndex = *handle & HR_HANDLE_MASK;  // extract the index

    // If there is only one slot left, and the gap is at maximum, the only session
    // context that we can safely load is the oldest one.
    if(s_oldestSavedSession < MAX_ACTIVE_SESSIONS && s_freeSessionSlots == 1
       && (CONTEXT_SLOT)gr.contextCounter == gr.contextArray[s_oldestSavedSession]
       && contextIndex != s_oldestSavedSession)
        return TPM_RC_CONTEXT_GAP;

    pAssert(contextIndex < MAX_ACTIVE_SESSIONS);

    // set the contextArray value to point to the session slot where
    // the context is loaded
    gr.contextArray[contextIndex] = slotIndex + 1;

    // if this was the oldest context, find the new oldest
    if(contextIndex == s_oldestSavedSession)
        ContextIdSetOldest();

    // Copy session data to session slot
    MemoryCopy(&s_sessions[slotIndex].session, session, sizeof(SESSION));

    // Set session slot as occupied
    s_sessions[slotIndex].occupied = TRUE;
```

```
    // Reduce the number of open spots
    s_freeSessionSlots--;

    return TPM_RC_SUCCESS;
}

//*** SessionFlush()
// This function is used to flush a session referenced by its handle.  If the
// session associated with 'handle' is loaded, the session array entry is
// marked as available.
//
// This function requires that 'handle' be a valid active session.
//
void SessionFlush(TPM_HANDLE handle  // IN: loaded or saved session handle
)
{
    CONTEXT_SLOT slotIndex;
    UINT32       contextIndex;  // Index into contextArray

    pAssert((HandleGetType(handle) == TPM_HT_POLICY_SESSION
             || HandleGetType(handle) == TPM_HT_HMAC_SESSION)
            && (SessionIsLoaded(handle) || SessionIsSaved(handle)));

    // Flush context ID of this session
    // Convert handle to an index into the contextArray
    contextIndex = handle & HR_HANDLE_MASK;

    pAssert(contextIndex < sizeof(gr.contextArray) / sizeof(gr.contextArray[0]));

    // Get the current contents of the array
    slotIndex = gr.contextArray[contextIndex];

    // Mark context array entry as available
    gr.contextArray[contextIndex] = 0;

    // Is this a saved session being flushed
    if(slotIndex > MAX_LOADED_SESSIONS)
    {
        // Flushing the oldest session?
        if(contextIndex == s_oldestSavedSession)
            // If so, find a new value for oldest.
            ContextIdSetOldest();
    }
    else
    {
        // Adjust slot index to point to session array index
        slotIndex -= 1;

        // Free session array index
        s_sessions[slotIndex].occupied = FALSE;
        s_freeSessionSlots++;
    }

    return;
}

//*** SessionComputeBoundEntity()
// This function computes the binding value for a session.  The binding value
// for a reserved handle is the handle itself.  For all the other entities,
// the authValue at the time of binding is included to prevent squatting.
// For those values, the Name and the authValue are concatenated
// into the bind buffer.  If they will not both fit, the will be overlapped
// by XORing bytes.  If XOR is required, the bind value will be full.
void SessionComputeBoundEntity(TPMI_DH_ENTITY entityHandle,  // IN: handle of entity
                               TPM2B_NAME*    bind           // OUT: binding value
)
```

```
{
    TPM2B_AUTH auth;
    BYTE*      pAuth = auth.t.buffer;
    UINT16     i;

    // Get name
    EntityGetName(entityHandle, bind);

    //    // The bound value of a reserved handle is the handle itself
    //    if(bind->t.size == sizeof(TPM_HANDLE)) return;

    // For all the other entities, concatenate the authorization value to the name.
    // Get a local copy of the authorization value because some overlapping
    // may be necessary.
    EntityGetAuthValue(entityHandle, &auth);

    // Make sure that the extra space is zeroed
    MemorySet(&bind->t.name[bind->t.size], 0, sizeof(bind->t.name) - bind->t.size);
    // XOR the authValue at the end of the name
    for(i = sizeof(bind->t.name) - auth.t.size; i < sizeof(bind->t.name); i++)
        bind->t.name[i] ^= *pAuth++;

    // Set the bind value to the maximum size
    bind->t.size = sizeof(bind->t.name);

    return;
}

//*** SessionSetStartTime()
// This function is used to initialize the session timing
void SessionSetStartTime(SESSION* session  // IN: the session to update
)
{
    session->startTime = g_time;
    session->epoch     = g_timeEpoch;
    session->timeout   = 0;
}

//*** SessionResetPolicyData()
// This function is used to reset the policy data without changing the nonce
// or the start time of the session.
void SessionResetPolicyData(SESSION* session  // IN: the session to reset
)
{
    SESSION_ATTRIBUTES oldAttributes;
    pAssert(session != NULL);

    // Will need later
    oldAttributes = session->attributes;

    // No command
    session->commandCode = 0;

    // No locality selected
    MemorySet(&session->commandLocality, 0, sizeof(session->commandLocality));

    // The cpHash size to zero
    session->u1.cpHash.b.size = 0;

    // No timeout
    session->timeout = 0;

    // Reset the pcrCounter
    session->pcrCounter = 0;

    // Reset the policy hash
```

```
    MemorySet(&session->u2.policyDigest.t.buffer, 0, session->u2.policyDigest.t.size);

    // Reset the session attributes
    MemorySet(&session->attributes, 0, sizeof(SESSION_ATTRIBUTES));

    // Restore the policy attributes
    session->attributes.isPolicy      = SET;
    session->attributes.isTrialPolicy = oldAttributes.isTrialPolicy;

    // Restore the bind attributes
    session->attributes.isDaBound      = oldAttributes.isDaBound;
    session->attributes.isLockoutBound = oldAttributes.isLockoutBound;
}

//*** SessionCapGetLoaded()
// This function returns a list of handles of loaded session, started
// from input 'handle'
//
// 'Handle' must be in valid loaded session handle range, but does not
// have to point to a loaded session.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
SessionCapGetLoaded(TPMI_SH_POLICY handle,     // IN: start handle
                    UINT32         count,      // IN: count of returned handles
                    TPML_HANDLE*   handleList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;

    pAssert(HandleGetType(handle) == TPM_HT_LOADED_SESSION);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES)
        count = MAX_CAP_HANDLES;

    // Iterate session context ID slots to get loaded session handles
    for(i = handle & HR_HANDLE_MASK; i < MAX_ACTIVE_SESSIONS; i++)
    {
        // If session is active
        if(gr.contextArray[i] != 0)
        {
            // If session is loaded
            if(gr.contextArray[i] <= MAX_LOADED_SESSIONS)
            {
                if(handleList->count < count)
                {
                    SESSION* session;

                    // If we have not filled up the return list, add this
                    // session handle to it
                    // assume that this is going to be an HMAC session
                    handle  = i + HMAC_SESSION_FIRST;
                    session = SessionGet(handle);
                    if(session->attributes.isPolicy)
                        handle = i + POLICY_SESSION_FIRST;
                    handleList->handle[handleList->count] = handle;
                    handleList->count++;
                }
                else
                {
```

```
                    // If the return list is full but we still have loaded object
                    // available, report this and stop iterating
                    more = YES;
                    break;
                }
            }
        }
    }

    return more;
}

//*** SessionCapGetOneLoaded()
// This function returns whether a session handle exists and is loaded.
BOOL SessionCapGetOneLoaded(TPMI_SH_POLICY handle)  // IN: handle
{
    pAssert(HandleGetType(handle) == TPM_HT_LOADED_SESSION);

    if((handle & HR_HANDLE_MASK) < MAX_ACTIVE_SESSIONS
       && gr.contextArray[(handle & HR_HANDLE_MASK)])
    {
        return TRUE;
    }

    return FALSE;
}

//*** SessionCapGetSaved()
// This function returns a list of handles for saved session, starting at
// 'handle'.
//
// 'Handle' must be in a valid handle range, but does not have to point to a
// saved session
//
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
SessionCapGetSaved(TPMI_SH_HMAC handle,     // IN: start handle
                   UINT32       count,      // IN: count of returned handles
                   TPML_HANDLE* handleList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;

    pAssert(HandleGetType(handle) == TPM_HT_SAVED_SESSION);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES)
        count = MAX_CAP_HANDLES;

    // Iterate session context ID slots to get loaded session handles
    for(i = handle & HR_HANDLE_MASK; i < MAX_ACTIVE_SESSIONS; i++)
    {
        // If session is active
        if(gr.contextArray[i] != 0)
        {
            // If session is saved
            if(gr.contextArray[i] > MAX_LOADED_SESSIONS)
            {
                if(handleList->count < count)
                {
```

```
                    // If we have not filled up the return list, add this
                    // session handle to it
                    handleList->handle[handleList->count] = i + HMAC_SESSION_FIRST;
                    handleList->count++;
                }
                else
                {
                    // If the return list is full but we still have loaded object
                    // available, report this and stop iterating
                    more = YES;
                    break;
                }
            }
        }
    }

    return more;
}

//*** SessionCapGetOneSaved()
// This function returns whether a session handle exists and is saved.
BOOL SessionCapGetOneSaved(TPMI_SH_HMAC handle)  // IN: handle
{
    pAssert(HandleGetType(handle) == TPM_HT_SAVED_SESSION);

    if((handle & HR_HANDLE_MASK) < MAX_ACTIVE_SESSIONS
       && gr.contextArray[(handle & HR_HANDLE_MASK)])
    {
        return TRUE;
    }

    return FALSE;
}

//*** SessionCapGetLoadedNumber()
// This function return the number of authorization sessions currently
// loaded into TPM RAM.
UINT32
SessionCapGetLoadedNumber(void)
{
    return MAX_LOADED_SESSIONS - s_freeSessionSlots;
}

//*** SessionCapGetLoadedAvail()
// This function returns the number of additional authorization sessions, of
// any type, that could be loaded into TPM RAM.
// NOTE: In other implementations, this number may just be an estimate. The only
//       requirement for the estimate is, if it is one or more, then at least one
//       session must be loadable.
UINT32
SessionCapGetLoadedAvail(void)
{
    return s_freeSessionSlots;
}

//*** SessionCapGetActiveNumber()
// This function returns the number of active authorization sessions currently
// being tracked by the TPM.
UINT32
SessionCapGetActiveNumber(void)
{
    UINT32 i;
    UINT32 num = 0;

    // Iterate the context array to find the number of non-zero slots
    for(i = 0; i < MAX_ACTIVE_SESSIONS; i++)
```

```
    {
        if(gr.contextArray[i] != 0)
            num++;
    }

    return num;
}

//*** SessionCapGetActiveAvail()
// This function returns the number of additional authorization sessions, of any
// type, that could be created. This not the number of slots for sessions, but
// the number of additional sessions that the TPM is capable of tracking.
UINT32
SessionCapGetActiveAvail(void)
{
    UINT32 i;
    UINT32 num = 0;

    // Iterate the context array to find the number of zero slots
    for(i = 0; i < MAX_ACTIVE_SESSIONS; i++)
    {
        if(gr.contextArray[i] == 0)
            num++;
    }

    return num;
}

//*** IsCpHashUnionOccupied()
// This function indicates whether the session attributes indicate that one of
// the members of the union containing `cpHash` are set.
BOOL IsCpHashUnionOccupied(SESSION_ATTRIBUTES attrs)
{
    return attrs.isBound || attrs.isCpHashDefined || attrs.isNameHashDefined
           || attrs.isParametersHashDefined || attrs.isTemplateHashDefined;
}
```

7.179 /tpm/src/subsystem/Time.c

```
//** Introduction
// This file contains the functions relating to the TPM's time functions including
// the interface to the implementation-specific time functions.
//
//** Includes
#include "Tpm.h"
#include "Marshal.h"

//** Functions

//*** TimePowerOn()
// This function initialize time info at _TPM_Init().
//
// This function is called at _TPM_Init() so that the TPM time can start counting
// as soon as the TPM comes out of reset and doesn't have to wait until
// TPM2_Startup() in order to begin the new time epoch. This could be significant
// for systems that could get powered up but not run any TPM commands for some
// period of time.
//
void TimePowerOn(void)
{
    g_time = _plat__TimerRead();
}

//*** TimeNewEpoch()
// This function does the processing to generate a new time epoch nonce and
```

```
// set NV for update. This function is only called when NV is known to be available
// and the clock is running. The epoch is updated to persistent data.
static void TimeNewEpoch(void)
{
#if CLOCK_STOPS
    CryptRandomGenerate(sizeof(CLOCK_NONCE), (BYTE*)&g_timeEpoch);
#else
    // if the epoch is kept in NV, update it.
    gp.timeEpoch++;
    NV_SYNC_PERSISTENT(timeEpoch);
#endif
    // Clean out any lingering state
    _plat__TimerWasStopped();
}

//*** TimeStartup()
// This function updates the resetCount and restartCount components of
// TPMS_CLOCK_INFO structure at TPM2_Startup().
//
// This function will deal with the deferred creation of a new epoch.
// TimeUpdateToCurrent() will not start a new epoch even if one is due when
// TPM_Startup() has not been run. This is because the state of NV is not known
// until startup completes. When Startup is done, then it will create the epoch
// nonce to complete the initializations by calling this function.
BOOL TimeStartup(STARTUP_TYPE type  // IN: start up type
)
{
    NOT_REFERENCED(type);
    // If the previous cycle is orderly shut down, the value of the safe bit
    // the same as previously saved.  Otherwise, it is not safe.
    if(!NV_IS_ORDERLY)
        go.clockSafe = NO;
    return TRUE;
}

//*** TimeClockUpdate()
// This function updates go.clock. If 'newTime' requires an update of NV, then
// NV is checked for availability. If it is not available or is rate limiting, then
// go.clock is not updated and the function returns an error. If 'newTime' would
// not cause an NV write, then go.clock is updated. If an NV write occurs, then
// go.safe is SET.
void TimeClockUpdate(UINT64 newTime  // IN: New time value in mS.
)
{
#define CLOCK_UPDATE_MASK ((1ULL << NV_CLOCK_UPDATE_INTERVAL) - 1)

    // Check to see if the update will cause a need for an nvClock update
    if((newTime | CLOCK_UPDATE_MASK) > (go.clock | CLOCK_UPDATE_MASK))
    {
        pAssert(g_NvStatus == TPM_RC_SUCCESS);

        // Going to update the NV time state so SET the safe flag
        go.clockSafe = YES;

        // update the time
        go.clock = newTime;

        NvWrite(NV_ORDERLY_DATA, sizeof(go), &go);
    }
    else
        // No NV update needed so just update
        go.clock = newTime;
}

//*** TimeUpdate()
// This function is used to update the time and clock values. If the TPM
```

```
// has run TPM2_Startup(), this function is called at the start of each command.
// If the TPM has not run TPM2_Startup(), this is called from TPM2_Startup() to
// get the clock values initialized. It is not called on command entry because, in
// this implementation, the go structure is not read from NV until TPM2_Startup().
// The reason for this is that the initialization code (_TPM_Init()) may run before
// NV is accessible.
void TimeUpdate(void)
{
    UINT64 elapsed;
    //
    // Make sure that we consume the current _plat__TimerWasStopped() state.
    if(_plat__TimerWasStopped())
    {
        TimeNewEpoch();
    }
    // Get the difference between this call and the last time we updated the tick
    // timer.
    elapsed = _plat__TimerRead() - g_time;
    // Don't read +
    g_time += elapsed;

    // Don't need to check the result because it has to be success because have
    // already checked that NV is available.
    TimeClockUpdate(go.clock + elapsed);

    // Call self healing logic for dictionary attack parameters
    DASelfHeal();
}

//*** TimeUpdateToCurrent()
// This function updates the 'Time' and 'Clock' in the global
// TPMS_TIME_INFO structure.
//
// In this implementation, 'Time' and 'Clock' are updated at the beginning
// of each command and the values are unchanged for the duration of the
// command.
//
// Because 'Clock' updates may require a write to NV memory, 'Time' and 'Clock'
// are not allowed to advance if NV is not available. When clock is not advancing,
// any function that uses 'Clock' will fail and return TPM_RC_NV_UNAVAILABLE or
// TPM_RC_NV_RATE.
//
// This implementation does not do rate limiting. If the implementation does do
// rate limiting, then the 'Clock' update should not be inhibited even when doing
// rate limiting.
void TimeUpdateToCurrent(void)
{
    // Can't update time during the dark interval or when rate limiting so don't
    // make any modifications to the internal clock value. Also, defer any clock
    // processing until TPM has run TPM2_Startup()
    if(!NV_IS_AVAILABLE || !TPMIsStarted())
        return;

    TimeUpdate();
}

//*** TimeSetAdjustRate()
// This function is used to perform rate adjustment on 'Time' and 'Clock'.
void TimeSetAdjustRate(TPM_CLOCK_ADJUST adjust  // IN: adjust constant
)
{
    switch(adjust)
    {
        case TPM_CLOCK_COARSE_SLOWER:
            _plat__ClockRateAdjust(PLAT_TPM_CLOCK_ADJUST_COARSE_SLOWER);
            break;
```

```
        case TPM_CLOCK_COARSE_FASTER:
            _plat__ClockRateAdjust(PLAT_TPM_CLOCK_ADJUST_COARSE_FASTER);
            break;
        case TPM_CLOCK_MEDIUM_SLOWER:
            _plat__ClockRateAdjust(PLAT_TPM_CLOCK_ADJUST_MEDIUM_SLOWER);
            break;
        case TPM_CLOCK_MEDIUM_FASTER:
            _plat__ClockRateAdjust(PLAT_TPM_CLOCK_ADJUST_MEDIUM_FASTER);
            break;
        case TPM_CLOCK_FINE_SLOWER:
            _plat__ClockRateAdjust(PLAT_TPM_CLOCK_ADJUST_FINE_SLOWER);
            break;
        case TPM_CLOCK_FINE_FASTER:
            _plat__ClockRateAdjust(PLAT_TPM_CLOCK_ADJUST_FINE_FASTER);
            break;
        case TPM_CLOCK_NO_CHANGE:
            break;
        default:
            // should have been blocked sooner
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }

    return;
}

//*** TimeGetMarshaled()
// This function is used to access TPMS_TIME_INFO in canonical form.
// The function collects the time information and marshals it into 'dataBuffer'
// and returns the marshaled size
UINT16
TimeGetMarshaled(TIME_INFO* dataBuffer  // OUT: result buffer
)
{
    TPMS_TIME_INFO timeInfo;

    // Fill TPMS_TIME_INFO structure
    timeInfo.time = g_time;
    TimeFillInfo(&timeInfo.clockInfo);

    // Marshal TPMS_TIME_INFO to canonical form
    return TPMS_TIME_INFO_Marshal(&timeInfo, (BYTE**)&dataBuffer, NULL);
}

//*** TimeFillInfo
// This function gathers information to fill in a TPMS_CLOCK_INFO structure.
void TimeFillInfo(TPMS_CLOCK_INFO* clockInfo)
{
    clockInfo->clock        = go.clock;
    clockInfo->resetCount   = gp.resetCount;
    clockInfo->restartCount = gr.restartCount;

    // If NV is not available, clock stopped advancing and the value reported is
    // not "safe".
    if(NV_IS_AVAILABLE)
        clockInfo->safe = go.clockSafe;
    else
        clockInfo->safe = NO;

    return;
}
```

7.180 /tpm/src/support/AlgorithmCap.c

```
//** Description
```

```
// This file contains the algorithm property definitions for the algorithms and the
// code for the TPM2_GetCapability() to return the algorithm properties.

//** Includes and Defines

#include "Tpm.h"

typedef struct
{
    TPM_ALG_ID     algID;
    TPMA_ALGORITHM attributes;
} ALGORITHM;

static const ALGORITHM s_algorithms[] = {
// The entries in this table need to be in ascending order but the table doesn't
// need to be full (gaps are allowed). One day, a tool might exist to fill in the
// table from the TPM_ALG description
#if ALG_RSA
    {TPM_ALG_RSA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 1, 0, 0, 0, 0, 0)},
#endif
#if ALG_SHA1
    {TPM_ALG_SHA1, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
#endif

    {TPM_ALG_HMAC, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 1, 0, 0, 0)},

#if ALG_AES
    {TPM_ALG_AES, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 0, 0, 0)},
#endif
#if ALG_MGF1
    {TPM_ALG_MGF1, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 1, 0)},
#endif

    {TPM_ALG_KEYEDHASH, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 1, 0, 1, 1, 0, 0)},

#if ALG_XOR
    {TPM_ALG_XOR, TPMA_ALGORITHM_INITIALIZER(0, 1, 1, 0, 0, 0, 0, 0, 0)},
#endif

#if ALG_SHA256
    {TPM_ALG_SHA256, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
#endif
#if ALG_SHA384
    {TPM_ALG_SHA384, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
#endif
#if ALG_SHA512
    {TPM_ALG_SHA512, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
#endif
#if ALG_SM3_256
    {TPM_ALG_SM3_256, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
#endif
#if ALG_SM4
    {TPM_ALG_SM4, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 0, 0, 0)},
#endif
#if ALG_RSASSA
    {TPM_ALG_RSASSA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif
#if ALG_RSAES
    {TPM_ALG_RSAES, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif
#if ALG_RSAPSS
    {TPM_ALG_RSAPSS, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif
#if ALG_OAEP
    {TPM_ALG_OAEP, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif
```

```
#if ALG_ECDSA
    {TPM_ALG_ECDSA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif
#if ALG_ECDH
    {TPM_ALG_ECDH, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 0, 0, 1, 0)},
#endif
#if ALG_ECDAA
    {TPM_ALG_ECDAA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif
#if ALG_SM2
    {TPM_ALG_SM2, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 1, 0)},
#endif
#if ALG_ECSCHNORR
    {TPM_ALG_ECSCHNORR, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif
#if ALG_ECMQV
    {TPM_ALG_ECMQV, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 0, 0, 1, 0)},
#endif
#if ALG_KDF1_SP800_56A
    {TPM_ALG_KDF1_SP800_56A, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 1, 0)},
#endif
#if ALG_KDF2
    {TPM_ALG_KDF2, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 1, 0)},
#endif
#if ALG_KDF1_SP800_108
    {TPM_ALG_KDF1_SP800_108, TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 1, 0)},
#endif
#if ALG_ECC
    {TPM_ALG_ECC, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 1, 0, 0, 0, 0, 0)},
#endif

    {TPM_ALG_SYMCIPHER, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 1, 0, 0, 0, 0, 0)},

#if ALG_CAMELLIA
    {TPM_ALG_CAMELLIA, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 0, 0, 0)},
#endif
#if ALG_CMAC
    {TPM_ALG_CMAC, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 1, 0, 0, 0)},
#endif
#if ALG_CTR
    {TPM_ALG_CTR, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 1, 0, 0)},
#endif
#if ALG_OFB
    {TPM_ALG_OFB, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 1, 0, 0)},
#endif
#if ALG_CBC
    {TPM_ALG_CBC, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 1, 0, 0)},
#endif
#if ALG_CFB
    {TPM_ALG_CFB, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 1, 0, 0)},
#endif
#if ALG_ECB
    {TPM_ALG_ECB, TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 1, 0, 0)},
#endif
};

//** AlgorithmCapGetImplemented()
// This function is used by TPM2_GetCapability() to return a list of the
// implemented algorithms.
//
//  Return Type: TPMI_YES_NO
//  YES         more algorithms to report
//  NO          no more algorithms to report
TPMI_YES_NO
AlgorithmCapGetImplemented(TPM_ALG_ID algID,  // IN: the starting algorithm ID
                           UINT32     count,  // IN: count of returned algorithms
```

```
                        TPML_ALG_PROPERTY* algList  // OUT: algorithm list
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;
    UINT32      algNum;

    // initialize output algorithm list
    algList->count = 0;

    // The maximum count of algorithms we may return is MAX_CAP_ALGS.
    if(count > MAX_CAP_ALGS)
        count = MAX_CAP_ALGS;

    // Compute how many algorithms are defined in s_algorithms array.
    algNum = sizeof(s_algorithms) / sizeof(s_algorithms[0]);

    // Scan the implemented algorithm list to see if there is a match to 'algID'.
    for(i = 0; i < algNum; i++)
    {
        // If algID is less than the starting algorithm ID, skip it
        if(s_algorithms[i].algID < algID)
            continue;
        if(algList->count < count)
        {
            // If we have not filled up the return list, add more algorithms
            // to it
            algList->algProperties[algList->count].alg = s_algorithms[i].algID;
            algList->algProperties[algList->count].algProperties =
                s_algorithms[i].attributes;
            algList->count++;
        }
        else
        {
            // If the return list is full but we still have algorithms
            // available, report this and stop scanning.
            more = YES;
            break;
        }
    }

    return more;
}

//** AlgorithmCapGetOneImplemented()
// This function returns whether a single algorithm was implemented, along
// with its properties (if implemented).
BOOL AlgorithmCapGetOneImplemented(
    TPM_ALG_ID         algID,       // IN: the algorithm ID
    TPMS_ALG_PROPERTY* algProperty  // OUT: algorithm properties
)
{
    UINT32 i;
    UINT32 algNum;

    // Compute how many algorithms are defined in s_algorithms array.
    algNum = sizeof(s_algorithms) / sizeof(s_algorithms[0]);

    // Scan the implemented algorithm list to see if there is a match to 'algID'.
    for(i = 0; i < algNum; i++)
    {
        // If algID is less than the starting algorithm ID, skip it
        if(s_algorithms[i].algID == algID)
        {
            algProperty->alg           = algID;
            algProperty->algProperties = s_algorithms[i].attributes;
```

```
            return TRUE;
        }
    }

    return FALSE;
}

//** AlgorithmGetImplementedVector()
// This function returns the bit vector of the implemented algorithms.
LIB_EXPORT
void AlgorithmGetImplementedVector(
    ALGORITHM_VECTOR* implemented  // OUT: the implemented bits are SET
)
{
    int index;

    // Nothing implemented until we say it is
    MemorySet(implemented, 0, sizeof(ALGORITHM_VECTOR));
    // Go through the list of implemented algorithms and SET the corresponding bit in
    // in the implemented vector
    for(index = (sizeof(s_algorithms) / sizeof(s_algorithms[0])) - 1; index >= 0;
        index--)
        SET_BIT(s_algorithms[index].algID, *implemented);
    return;
}
```

7.181 /tpm/src/support/Bits.c

```
//** Introduction
// This file contains bit manipulation routines.  They operate on bit arrays.
//
// The 0th bit in the array is the right-most bit in the 0th octet in
// the array.
//
// NOTE: If pAssert() is defined, the functions will assert if the indicated bit
// number is outside of the range of 'bArray'. How the assert is handled is
// implementation dependent.

//** Includes

#include "Tpm.h"

//** Functions

//*** TestBit()
// This function is used to check the setting of a bit in an array of bits.
//  Return Type: BOOL
//      TRUE(1)         bit is set
//      FALSE(0)        bit is not set
BOOL TestBit(unsigned int bitNum,       // IN: number of the bit in 'bArray'
             BYTE*        bArray,       // IN: array containing the bits
             unsigned int bytesInArray  // IN: size in bytes of 'bArray'
)
{
    pAssert(bytesInArray > (bitNum >> 3));
    return ((bArray[bitNum >> 3] & (1 << (bitNum & 7))) != 0);
}

//*** SetBit()
// This function will set the indicated bit in 'bArray'.
void SetBit(unsigned int bitNum,       // IN: number of the bit in 'bArray'
            BYTE*        bArray,       // IN: array containing the bits
            unsigned int bytesInArray  // IN: size in bytes of 'bArray'
)
{
```

```
    pAssert(bytesInArray > (bitNum >> 3));
    bArray[bitNum >> 3] |= (1 << (bitNum & 7));
}

//*** ClearBit()
// This function will clear the indicated bit in 'bArray'.
void ClearBit(unsigned int bitNum,       // IN: number of the bit in 'bArray'.
              BYTE*        bArray,       // IN: array containing the bits
              unsigned int bytesInArray  // IN: size in bytes of 'bArray'
)
{
    pAssert(bytesInArray > (bitNum >> 3));
    bArray[bitNum >> 3] &= ~(1 << (bitNum & 7));
}
```

7.182 /tpm/src/support/CommandCodeAttributes.c

```
//** Introduction
// This file contains the functions for testing various command properties.

//** Includes and Defines

#include "Tpm.h"
#include "CommandCodeAttributes_fp.h"

// Set the default value for CC_VEND if not already set
#ifndef CC_VEND
#  define CC_VEND (TPM_CC)(0x20000000)
#endif

typedef UINT16 ATTRIBUTE_TYPE;

// The following file is produced from the command tables in part 3 of the
// specification. It defines the attributes for each of the commands.
// NOTE: This file is currently produced by an automated process. Files
// produced from Part 2 or Part 3 tables through automated processes are not
// included in the specification so that their is no ambiguity about the
// table containing the information being the normative definition.
#define _COMMAND_CODE_ATTRIBUTES_
#include "CommandAttributeData.h"

//** Command Attribute Functions

//*** NextImplementedIndex()
// This function is used when the lists are not compressed. In a compressed list,
// only the implemented commands are present. So, a search might find a value
// but that value may not be implemented. This function checks to see if the input
// commandIndex points to an implemented command and, if not, it searches upwards
// until it finds one. When the list is compressed, this function gets defined
// as a no-op.
//  Return Type: COMMAND_INDEX
//  UNIMPLEMENTED_COMMAND_INDEX     command is not implemented
//  other                           index of the command
#if !COMPRESSED_LISTS
static COMMAND_INDEX NextImplementedIndex(COMMAND_INDEX commandIndex)
{
    for(; commandIndex < COMMAND_COUNT; commandIndex++)
    {
        if(s_commandAttributes[commandIndex] & IS_IMPLEMENTED)
            return commandIndex;
    }
    return UNIMPLEMENTED_COMMAND_INDEX;
}
#else
#  define NextImplementedIndex(x) (x)
```

```
#endif

//*** GetClosestCommandIndex()
// This function returns the command index for the command with a value that is
// equal to or greater than the input value
//  Return Type: COMMAND_INDEX
//  UNIMPLEMENTED_COMMAND_INDEX     command is not implemented
//  other                           index of a command
COMMAND_INDEX
GetClosestCommandIndex(TPM_CC commandCode  // IN: the command code to start at
)
{
    BOOL          vendor      = (commandCode & CC_VEND) != 0;
    COMMAND_INDEX searchIndex = (COMMAND_INDEX)commandCode;

    // The commandCode is a UINT32 and the search index is UINT16. We are going to
    // search for a match but need to make sure that the commandCode value is not
    // out of range. To do this, need to clear the vendor bit of the commandCode
    // (if set) and compare the result to the 16-bit searchIndex value. If it is
    // out of range, indicate that the command is not implemented
    if((commandCode & ~CC_VEND) != searchIndex)
        return UNIMPLEMENTED_COMMAND_INDEX;

    // if there is at least one vendor command, the last entry in the array will
    // have the v bit set. If the input commandCode is larger than the last
    // vendor-command, then it is out of range.
    if(vendor)
    {
#if VENDOR_COMMAND_ARRAY_SIZE > 0
        COMMAND_INDEX commandIndex;
        COMMAND_INDEX min;
        COMMAND_INDEX max;
        int           diff;
#  if LIBRARY_COMMAND_ARRAY_SIZE == COMMAND_COUNT
#    error "Constants are not consistent."
#  endif
        // Check to see if the value is equal to or below the minimum
        // entry.
        // Note: Put this check first so that the typical case of only one vendor-
        // specific command doesn't waste any more time.
        if(GET_ATTRIBUTE(s_ccAttr[LIBRARY_COMMAND_ARRAY_SIZE], TPMA_CC, commandIndex)
           >= searchIndex)
        {
            // the vendor array is always assumed to be packed so there is
            // no need to check to see if the command is implemented
            return LIBRARY_COMMAND_ARRAY_SIZE;
        }
        // See if this is out of range on the top
        if(GET_ATTRIBUTE(s_ccAttr[COMMAND_COUNT - 1], TPMA_CC, commandIndex)
           < searchIndex)
        {
            return UNIMPLEMENTED_COMMAND_INDEX;
        }
        commandIndex = UNIMPLEMENTED_COMMAND_INDEX;  // Needs initialization to keep
                                                     // compiler happy
        min  = LIBRARY_COMMAND_ARRAY_SIZE;           // first vendor command
        max  = COMMAND_COUNT - 1;                    // last vendor command
        diff = 1;                                    // needs initialization to keep
                                                     // compiler happy
        while(min <= max)
        {
            commandIndex = (min + max + 1) / 2;
            diff = GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex)
                   - searchIndex;
            if(diff == 0)
                return commandIndex;
```

```
            if(diff > 0)
                max = commandIndex - 1;
            else
                min = commandIndex + 1;
        }
        // didn't find and exact match. commandIndex will be pointing at the last
        // item tested. If 'diff' is positive, then the last item tested was
        // larger index of the command code so it is the smallest value
        // larger than the requested value.
        if(diff > 0)
            return commandIndex;
        // if 'diff' is negative, then the value tested was smaller than
        // the commandCode index and the next higher value is the correct one.
        // Note: this will necessarily be in range because of the earlier check
        // that the index was within range.
        return commandIndex + 1;
#else
        // If there are no vendor commands so anything with the vendor bit set is out
        // of range
        return UNIMPLEMENTED_COMMAND_INDEX;
#endif
    }
    // Get here if the V-Bit was not set in 'commandCode'

    if(GET_ATTRIBUTE(s_ccAttr[LIBRARY_COMMAND_ARRAY_SIZE - 1], TPMA_CC, commandIndex)
       < searchIndex)
    {
        // requested index is out of the range to the top
#if VENDOR_COMMAND_ARRAY_SIZE > 0
        // If there are vendor commands, then the first vendor command
        // is the next value greater than the commandCode.
        // NOTE: we got here if the starting index did not have the V bit but we
        // reached the end of the array of library commands (non-vendor). Since
        // there is at least one vendor command, and vendor commands are always
        // in a compressed list that starts after the library list, the next
        // index value contains a valid vendor command.
        return LIBRARY_COMMAND_ARRAY_SIZE;
#else
        // if there are no vendor commands, then this is out of range
        return UNIMPLEMENTED_COMMAND_INDEX;
#endif
    }
    // If the request is lower than any value in the array, then return
    // the lowest value (needs to be an index for an implemented command
    if(GET_ATTRIBUTE(s_ccAttr[0], TPMA_CC, commandIndex) >= searchIndex)
    {
        return NextImplementedIndex(0);
    }
    else
    {
#if COMPRESSED_LISTS
        COMMAND_INDEX commandIndex = UNIMPLEMENTED_COMMAND_INDEX;
        COMMAND_INDEX min          = 0;
        COMMAND_INDEX max          = LIBRARY_COMMAND_ARRAY_SIZE - 1;
        int           diff         = 1;
#  if LIBRARY_COMMAND_ARRAY_SIZE == 0
#    error "Something is terribly wrong"
#  endif
        // The s_ccAttr array contains an extra entry at the end (a zero value).
        // Don't count this as an array entry. This means that max should start
        // out pointing to the last valid entry in the array which is - 2
        pAssert(
            max
            == (sizeof(s_ccAttr) / sizeof(TPMA_CC) - VENDOR_COMMAND_ARRAY_SIZE - 2));
        while(min <= max)
        {
```

```
            commandIndex = (min + max + 1) / 2;
            diff = GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex)
                   - searchIndex;
            if(diff == 0)
                return commandIndex;
            if(diff > 0)
                max = commandIndex - 1;
            else
                min = commandIndex + 1;
        }
        // didn't find and exact match. commandIndex will be pointing at the
        // last item tested. If diff is positive, then the last item tested was
        // larger index of the command code so it is the smallest value
        // larger than the requested value.
        if(diff > 0)
            return commandIndex;
        // if diff is negative, then the value tested was smaller than
        // the commandCode index and the next higher value is the correct one.
        // Note: this will necessarily be in range because of the earlier check
        // that the index was within range.
        return commandIndex + 1;
#else
        // The list is not compressed so offset into the array by the command
        // code value of the first entry in the list. Then go find the first
        // implemented command.
        return NextImplementedIndex(
            searchIndex - (COMMAND_INDEX)s_ccAttr[0].commandIndex);
#endif
    }
}

//*** CommandCodeToComandIndex()
// This function returns the index in the various attributes arrays of the
// command.
//  Return Type: COMMAND_INDEX
//  UNIMPLEMENTED_COMMAND_INDEX     command is not implemented
//  other                           index of the command
COMMAND_INDEX
CommandCodeToCommandIndex(TPM_CC commandCode  // IN: the command code to look up
)
{
    // Extract the low 16-bits of the command code to get the starting search index
    COMMAND_INDEX searchIndex = (COMMAND_INDEX)commandCode;
    BOOL          vendor      = (commandCode & CC_VEND) != 0;
    COMMAND_INDEX commandIndex;
#if !COMPRESSED_LISTS
    if(!vendor)
    {
        commandIndex = searchIndex - (COMMAND_INDEX)s_ccAttr[0].commandIndex;
        // Check for out of range or unimplemented.
        // Note, since a COMMAND_INDEX is unsigned, if searchIndex is smaller than
        // the lowest value of command, it will become a 'negative' number making
        // it look like a large unsigned number, this will cause it to fail
        // the unsigned check below.
        if(commandIndex >= LIBRARY_COMMAND_ARRAY_SIZE
           || (s_commandAttributes[commandIndex] & IS_IMPLEMENTED) == 0)
            return UNIMPLEMENTED_COMMAND_INDEX;
        return commandIndex;
    }
#endif
    // Need this code for any vendor code lookup or for compressed lists
    commandIndex = GetClosestCommandIndex(commandCode);

    // Look at the returned value from get closest. If it isn't the one that was
    // requested, then the command is not implemented.
    if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
```

```
    {
        if((GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex)
            != searchIndex)
           || (IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V)) != vendor)
            commandIndex = UNIMPLEMENTED_COMMAND_INDEX;
    }
    return commandIndex;
}

//*** GetNextCommandIndex()
// This function returns the index of the next implemented command.
//  Return Type: COMMAND_INDEX
//  UNIMPLEMENTED_COMMAND_INDEX     no more implemented commands
//  other                           the index of the next implemented command
COMMAND_INDEX
GetNextCommandIndex(COMMAND_INDEX commandIndex  // IN: the starting index
)
{
    while(++commandIndex < COMMAND_COUNT)
    {
#if !COMPRESSED_LISTS
        if(s_commandAttributes[commandIndex] & IS_IMPLEMENTED)
#endif
            return commandIndex;
    }
    return UNIMPLEMENTED_COMMAND_INDEX;
}

//*** GetCommandCode()
// This function returns the commandCode associated with the command index
TPM_CC
GetCommandCode(COMMAND_INDEX commandIndex  // IN: the command index
)
{
    TPM_CC commandCode = GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex);
    if(IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))
        commandCode += CC_VEND;
    return commandCode;
}

//*** CommandAuthRole()
//
//  This function returns the authorization role required of a handle.
//
//  Return Type: AUTH_ROLE
//  AUTH_NONE       no authorization is required
//  AUTH_USER       user role authorization is required
//  AUTH_ADMIN      admin role authorization is required
//  AUTH_DUP        duplication role authorization is required
AUTH_ROLE
CommandAuthRole(COMMAND_INDEX commandIndex,  // IN: command index
                UINT32        handleIndex    // IN: handle index (zero based)
)
{
    if(0 == handleIndex)
    {
        // Any authorization role set?
        COMMAND_ATTRIBUTES properties = s_commandAttributes[commandIndex];

        if(properties & HANDLE_1_USER)
            return AUTH_USER;
        if(properties & HANDLE_1_ADMIN)
            return AUTH_ADMIN;
        if(properties & HANDLE_1_DUP)
            return AUTH_DUP;
    }
```

```
    else if(1 == handleIndex)
    {
        if(s_commandAttributes[commandIndex] & HANDLE_2_USER)
            return AUTH_USER;
    }
    return AUTH_NONE;
}

//*** EncryptSize()
// This function returns the size of the decrypt size field. This function returns
// 0 if encryption is not allowed
//  Return Type: int
//  0       encryption not allowed
//  2       size field is two bytes
//  4       size field is four bytes
int EncryptSize(COMMAND_INDEX commandIndex  // IN: command index
)
{
    return ((s_commandAttributes[commandIndex] & ENCRYPT_2)   ? 2
            : (s_commandAttributes[commandIndex] & ENCRYPT_4) ? 4
                                                             : 0);
}

//*** DecryptSize()
// This function returns the size of the decrypt size field. This function returns
// 0 if decryption is not allowed
//  Return Type: int
//  0       encryption not allowed
//  2       size field is two bytes
//  4       size field is four bytes
int DecryptSize(COMMAND_INDEX commandIndex  // IN: command index
)
{
    return ((s_commandAttributes[commandIndex] & DECRYPT_2)   ? 2
            : (s_commandAttributes[commandIndex] & DECRYPT_4) ? 4
                                                             : 0);
}

//*** IsSessionAllowed()
//
// This function indicates if the command is allowed to have sessions.
//
// This function must not be called if the command is not known to be implemented.
//
//  Return Type: BOOL
//      TRUE(1)         session is allowed with this command
//      FALSE(0)        session is not allowed with this command
BOOL IsSessionAllowed(COMMAND_INDEX commandIndex  // IN: the command to be checked
)
{
    return ((s_commandAttributes[commandIndex] & NO_SESSIONS) == 0);
}

//*** IsHandleInResponse()
// This function determines if a command has a handle in the response
BOOL IsHandleInResponse(COMMAND_INDEX commandIndex)
{
    return ((s_commandAttributes[commandIndex] & R_HANDLE) != 0);
}

//*** IsWriteOperation()
// Checks to see if an operation will write to an NV Index and is subject to being
// blocked by read-lock
BOOL IsWriteOperation(COMMAND_INDEX commandIndex  // IN: Command to check
)
{
```

```
#ifdef WRITE_LOCK
    return ((s_commandAttributes[commandIndex] & WRITE_LOCK) != 0);
#else
    if(!IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))
    {
        switch(GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex))
        {
            case TPM_CC_NV_Write:
#  if CC_NV_Increment
            case TPM_CC_NV_Increment:
#  endif
#  if CC_NV_SetBits
            case TPM_CC_NV_SetBits:
#  endif
#  if CC_NV_Extend
            case TPM_CC_NV_Extend:
#  endif
#  if CC_AC_Send
            case TPM_CC_AC_Send:
#  endif
            // NV write lock counts as a write operation for authorization purposes.
            // We check to see if the NV is write locked before we do the
            // authorization. If it is locked, we fail the command early.
            case TPM_CC_NV_WriteLock:
                return TRUE;
            default:
                break;
        }
    }
    return FALSE;
#endif
}

//*** IsReadOperation()
// Checks to see if an operation will write to an NV Index and is
// subject to being blocked by write-lock.
BOOL IsReadOperation(COMMAND_INDEX commandIndex  // IN: Command to check
)
{
#ifdef READ_LOCK
    return ((s_commandAttributes[commandIndex] & READ_LOCK) != 0);
#else

    if(!IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))
    {
        switch(GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex))
        {
            case TPM_CC_NV_Read:
            case TPM_CC_PolicyNV:
            case TPM_CC_NV_Certify:
            // NV read lock counts as a read operation for authorization purposes.
            // We check to see if the NV is read locked before we do the
            // authorization. If it is locked, we fail the command early.
            case TPM_CC_NV_ReadLock:
                return TRUE;
            default:
                break;
        }
    }
    return FALSE;
#endif
}

//*** CommandCapGetCCList()
// This function returns a list of implemented commands and command attributes
// starting from the command in 'commandCode'.
```

```
//  Return Type: TPMI_YES_NO
//      YES         more command attributes are available
//      NO          no more command attributes are available
TPMI_YES_NO
CommandCapGetCCList(TPM_CC commandCode,  // IN: start command code
                    UINT32 count,        // IN: maximum count for number of entries in
                                         //     'commandList'
                    TPML_CCA* commandList  // OUT: list of TPMA_CC
)
{
    TPMI_YES_NO   more = NO;
    COMMAND_INDEX commandIndex;

    // initialize output handle list count
    commandList->count = 0;

    for(commandIndex = GetClosestCommandIndex(commandCode);
        commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
        commandIndex = GetNextCommandIndex(commandIndex))
    {
#if !COMPRESSED_LISTS
        // this check isn't needed for compressed lists.
        if(!(s_commandAttributes[commandIndex] & IS_IMPLEMENTED))
            continue;
#endif
        if(commandList->count < count)
        {
            // If the list is not full, add the attributes for this command.
            commandList->commandAttributes[commandList->count] =
                s_ccAttr[commandIndex];
            commandList->count++;
        }
        else
        {
            // If the list is full but there are more commands to report,
            // indicate this and return.
            more = YES;
            break;
        }
    }
    return more;
}

//*** CommandCapGetOneCC()
// This function checks whether a command is implemented, and returns its
// attributes if so.
BOOL CommandCapGetOneCC(TPM_CC   commandCode,       // IN: command code
                        TPMA_CC* commandAttributes  // OUT: command attributes
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);
    if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
    {
        *commandAttributes = s_ccAttr[commandIndex];
        return TRUE;
    }
    return FALSE;
}

//*** IsVendorCommand()
// Function indicates if a command index references a vendor command.
//  Return Type: BOOL
//      TRUE(1)         command is a vendor command
//      FALSE(0)        command is not a vendor command
BOOL IsVendorCommand(COMMAND_INDEX commandIndex  // IN: command index to check
)
```

```
{
    return (IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V));
}
```

7.183 /tpm/src/support/Entity.c

```
//** Description
// The functions in this file are used for accessing properties for handles of
// various types. Functions in other files require handles of a specific
// type but the functions in this file allow use of any handle type.

//** Includes

#include "Tpm.h"

//** Functions
//*** EntityGetLoadStatus()
// This function will check that all the handles access loaded entities.
//  Return Type: TPM_RC
//      TPM_RC_HANDLE           handle type does not match
//      TPM_RC_REFERENCE_Hx     entity is not present
//      TPM_RC_HIERARCHY        entity belongs to a disabled hierarchy
//      TPM_RC_OBJECT_MEMORY    handle is an evict object but there is no
//                               space to load it to RAM
TPM_RC
EntityGetLoadStatus(COMMAND* command  // IN/OUT: command parsing structure
)
{
    UINT32 i;
    TPM_RC result = TPM_RC_SUCCESS;
    //
    for(i = 0; i < command->handleNum; i++)
    {
        TPM_HANDLE handle = command->handles[i];
        switch(HandleGetType(handle))
        {
            // For handles associated with hierarchies, the entity is present
            // only if the associated enable is SET.
            case TPM_HT_PERMANENT:
                switch(handle)
                {
                    // First handle non-hierarchy cases
#if VENDOR_PERMANENT_AUTH_ENABLED == YES
                    case VENDOR_PERMANENT_AUTH_HANDLE:
                        if(!gc.ehEnable)
                            result = TPM_RC_HIERARCHY;
                        break;
#endif
                    // PW session handle and lockout handle are always available
                    case TPM_RS_PW:
                        // Need to be careful for lockout. Lockout is always available
                        // for policy checks but not always available when authValue
                        // is being checked.
                    case TPM_RH_LOCKOUT:
                        // Rather than have #ifdefs all over the code,
                        // CASE_ACT_HANDLE is defined in ACT.h. It is 'case
TPM_RH_ACT_x:'
                        // FOR_EACH_ACT(CASE_ACT_HANDLE) creates a simple
                        // case TPM_RH_ACT_x: // for each of the implemented ACT.
                        FOR_EACH_ACT(CASE_ACT_HANDLE)
                        break;
                    default:
                        // If the implementation has a manufacturer-specific value
                        // then test for it here. Since this implementation does
                        // not have any, this implementation returns the same failure
```

```
                        // that unmarshaling of a bad handle would produce.
                        if(((TPM_RH)handle >= TPM_RH_AUTH_00)
                           && ((TPM_RH)handle <= TPM_RH_AUTH_FF))
                            // if the implementation has a manufacturer-specific value
                            result = TPM_RC_VALUE;
                        else
                            // The handle either refers to a hierarchy or is invalid.
                            result = ValidateHierarchy(handle);
                        break;
                    }
                    break;
                case TPM_HT_TRANSIENT:
                    // For a transient object, check if the handle is associated
                    // with a loaded object.
                    if(!IsObjectPresent(handle))
                        result = TPM_RC_REFERENCE_H0;
                    break;
                case TPM_HT_PERSISTENT:
                    // Persistent object
                    // Copy the persistent object to RAM and replace the handle with the
                    // handle of the assigned slot.  A TPM_RC_OBJECT_MEMORY,
                    // TPM_RC_HIERARCHY or TPM_RC_REFERENCE_H0 error may be returned by
                    // ObjectLoadEvict()
                    result = ObjectLoadEvict(&command->handles[i], command->index);
                    break;
                case TPM_HT_HMAC_SESSION:
                    // For an HMAC session, see if the session is loaded
                    // and if the session in the session slot is actually
                    // an HMAC session.
                    if(SessionIsLoaded(handle))
                    {
                        SESSION* session;
                        session = SessionGet(handle);
                        // Check if the session is a HMAC session
                        if(session->attributes.isPolicy == SET)
                            result = TPM_RC_HANDLE;
                    }
                    else
                        result = TPM_RC_REFERENCE_H0;
                    break;
                case TPM_HT_POLICY_SESSION:
                    // For a policy session, see if the session is loaded
                    // and if the session in the session slot is actually
                    // a policy session.
                    if(SessionIsLoaded(handle))
                    {
                        SESSION* session;
                        session = SessionGet(handle);
                        // Check if the session is a policy session
                        if(session->attributes.isPolicy == CLEAR)
                            result = TPM_RC_HANDLE;
                    }
                    else
                        result = TPM_RC_REFERENCE_H0;
                    break;
                case TPM_HT_NV_INDEX:
                    // For an NV Index, use the TPM-specific routine
                    // to search the IN Index space.
                    result = NvIndexIsAccessible(handle);
                    break;
                case TPM_HT_PCR:
                    // Any PCR handle that is unmarshaled successfully referenced
                    // a PCR that is defined.
                    break;
#if CC_AC_Send
                case TPM_HT_AC:
```

```
                // Use the TPM-specific routine to search for the AC
                result = AcIsAccessible(handle);
                break;
#endif
            case TPM_HT_EXTERNAL_NV:
            case TPM_HT_PERMANENT_NV:
                // Not yet supported.
                result = TPM_RC_VALUE;
                break;
            default:
                // Any other handle type is a defect in the unmarshaling code.
                FAIL(FATAL_ERROR_INTERNAL);
                break;
        }
        if(result != TPM_RC_SUCCESS)
        {
            if(result == TPM_RC_REFERENCE_H0)
                result = result + i;
            else
                result = RcSafeAddToResult(result, TPM_RC_H + g_rcIndex[i]);
            break;
        }
    }
    return result;
}

//*** EntityGetAuthValue()
// This function is used to access the 'authValue' associated with a handle.
// This function assumes that the handle references an entity that is accessible
// and the handle is not for a persistent objects. That is EntityGetLoadStatus()
// should have been called. Also, the accessibility of the authValue should have
// been verified by IsAuthValueAvailable().
//
// This function copies the authorization value of the entity to 'auth'.
// Return Type: UINT16
//      count           number of bytes in the authValue with 0's stripped
UINT16
EntityGetAuthValue(TPMI_DH_ENTITY handle,  // IN: handle of entity
                   TPM2B_AUTH*    auth     // OUT: authValue of the entity
)
{
    TPM2B_AUTH* pAuth = NULL;

    auth->t.size      = 0;

    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
        {
            switch(HierarchyNormalizeHandle(handle))
            {
                case TPM_RH_OWNER:
                    // ownerAuth for TPM_RH_OWNER
                    pAuth = &gp.ownerAuth;
                    break;
                case TPM_RH_ENDORSEMENT:
                    // endorsementAuth for TPM_RH_ENDORSEMENT
                    pAuth = &gp.endorsementAuth;
                    break;

                    // The ACT use platformAuth for auth
                    FOR_EACH_ACT(CASE_ACT_HANDLE)

                case TPM_RH_PLATFORM:
                    // platformAuth for TPM_RH_PLATFORM
                    pAuth = &gc.platformAuth;
```

```
                    break;
                case TPM_RH_LOCKOUT:
                    // lockoutAuth for TPM_RH_LOCKOUT
                    pAuth = &gp.lockoutAuth;
                    break;
                case TPM_RH_NULL:
                    // nullAuth for TPM_RH_NULL. Return 0 directly here
                    return 0;
                    break;
#if VENDOR_PERMANENT_AUTH_ENABLED == YES
                case VENDOR_PERMANENT_AUTH_HANDLE:
                    // vendor authorization value
                    pAuth = &g_platformUniqueAuth;
#endif
                default:
                    // If any other permanent handle is present it is
                    // a code defect.
                    FAIL(FATAL_ERROR_INTERNAL);
                    break;
            }
            break;
        }
        case TPM_HT_TRANSIENT:
            // authValue for an object
            // A persistent object would have been copied into RAM
            // and would have an transient object handle here.
            {
                OBJECT* object;

                object = HandleToObject(handle);
                // special handling if this is a sequence object
                if(ObjectIsSequence(object))
                {
                    pAuth = &((HASH_OBJECT*)object)->auth;
                }
                else
                {
                    // Authorization is available only when the private portion of
                    // the object is loaded.  The check should be made before
                    // this function is called
                    pAssert(object->attributes.publicOnly == CLEAR);
                    pAuth = &object->sensitive.authValue;
                }
            }
            break;
        case TPM_HT_NV_INDEX:
            // authValue for an NV index
            {
                NV_INDEX* nvIndex = NvGetIndexInfo(handle, NULL);
                pAssert(nvIndex != NULL);
                pAuth = &nvIndex->authValue;
            }
            break;
        case TPM_HT_PCR:
            // authValue for PCR
            pAuth = PCRGetAuthValue(handle);
            break;
        default:
            // If any other handle type is present here, then there is a defect
            // in the unmarshaling code.
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    // Copy the authValue
    MemoryCopy2B((TPM2B*)auth, (TPM2B*)pAuth, sizeof(auth->t.buffer));
    MemoryRemoveTrailingZeros(auth);
```

```
    return auth->t.size;
}

//*** EntityGetAuthPolicy()
// This function is used to access the 'authPolicy' associated with a handle.
// This function assumes that the handle references an entity that is accessible
// and the handle is not for a persistent objects. That is EntityGetLoadStatus()
// should have been called. Also, the accessibility of the authPolicy should have
// been verified by IsAuthPolicyAvailable().
//
// This function copies the authorization policy of the entity to 'authPolicy'.
//
//  The return value is the hash algorithm for the policy.
TPMI_ALG_HASH
EntityGetAuthPolicy(TPMI_DH_ENTITY handle,     // IN: handle of entity
                    TPM2B_DIGEST*  authPolicy  // OUT: authPolicy of the entity
)
{
    TPMI_ALG_HASH hashAlg = TPM_ALG_NULL;
    authPolicy->t.size    = 0;

    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
            switch(HierarchyNormalizeHandle(handle))
            {
                case TPM_RH_OWNER:
                    // ownerPolicy for TPM_RH_OWNER
                    *authPolicy = gp.ownerPolicy;
                    hashAlg     = gp.ownerAlg;
                    break;
                case TPM_RH_ENDORSEMENT:
                    // endorsementPolicy for TPM_RH_ENDORSEMENT
                    *authPolicy = gp.endorsementPolicy;
                    hashAlg     = gp.endorsementAlg;
                    break;
                case TPM_RH_PLATFORM:
                    // platformPolicy for TPM_RH_PLATFORM
                    *authPolicy = gc.platformPolicy;
                    hashAlg     = gc.platformAlg;
                    break;
                case TPM_RH_LOCKOUT:
                    // lockoutPolicy for TPM_RH_LOCKOUT
                    *authPolicy = gp.lockoutPolicy;
                    hashAlg     = gp.lockoutAlg;
                    break;
#define ACT_GET_POLICY(N)                    \
    case TPM_RH_ACT_##N:                     \
        *authPolicy = go.ACT_##N.authPolicy; \
        hashAlg     = go.ACT_##N.hashAlg;    \
        break;
                    // Get the policy for each implemented ACT
                    FOR_EACH_ACT(ACT_GET_POLICY)
                default:
                    hashAlg = TPM_ALG_ERROR;
                    break;
            }
            break;
        case TPM_HT_TRANSIENT:
            // authPolicy for an object
            {
                OBJECT* object = HandleToObject(handle);
                *authPolicy    = object->publicArea.authPolicy;
                hashAlg        = object->publicArea.nameAlg;
            }
            break;
```

```
        case TPM_HT_NV_INDEX:
            // authPolicy for a NV index
            {
                NV_INDEX* nvIndex = NvGetIndexInfo(handle, NULL);
                pAssert(nvIndex != 0);
                *authPolicy = nvIndex->publicArea.authPolicy;
                hashAlg     = nvIndex->publicArea.nameAlg;
            }
            break;
        case TPM_HT_PCR:
            // authPolicy for a PCR
            hashAlg = PCRGetAuthPolicy(handle, authPolicy);
            break;
        default:
            // If any other handle type is present it is a code defect.
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    return hashAlg;
}

//*** EntityGetName()
// This function returns the Name associated with a handle.
TPM2B_NAME* EntityGetName(TPMI_DH_ENTITY handle,  // IN: handle of entity
                          TPM2B_NAME*    name     // OUT: name of entity
)
{
    switch(HandleGetType(handle))
    {
        case TPM_HT_TRANSIENT:
        {
            // Name for an object
            OBJECT* object = HandleToObject(handle);
            // an object with no nameAlg has no name
            if(object->publicArea.nameAlg == TPM_ALG_NULL)
                name->b.size = 0;
            else
                *name = object->name;
            break;
        }
        case TPM_HT_NV_INDEX:
            // Name for a NV index
            NvGetNameByIndexHandle(handle, name);
            break;
        default:
            // For all other types, the handle is the Name
            name->t.size = sizeof(TPM_HANDLE);
            UINT32_TO_BYTE_ARRAY(handle, name->t.name);
            break;
    }
    return name;
}

//*** EntityGetHierarchy()
// This function returns the hierarchy handle associated with an entity.
// a) A handle that is a hierarchy handle is associated with itself.
// b) An NV index belongs to TPM_RH_PLATFORM if TPMA_NV_PLATFORMCREATE,
//    is SET, otherwise it belongs to TPM_RH_OWNER
// c) An object handle belongs to its hierarchy.
TPMI_RH_HIERARCHY
EntityGetHierarchy(TPMI_DH_ENTITY handle  // IN :handle of entity
)
{
    TPMI_RH_HIERARCHY hierarchy = TPM_RH_NULL;

    switch(HandleGetType(handle))
```

```
{
    case TPM_HT_PERMANENT:
        // hierarchy for a permanent handle

        if(HierarchyIsFirmwareLimited(handle) || HierarchyIsSvnLimited(handle))
        {
            hierarchy = handle;
            break;
        }

        switch(handle)
        {
            case TPM_RH_PLATFORM:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_NULL:
                hierarchy = handle;
                break;
            // all other permanent handles are associated with the owner
            // hierarchy.  (should only be TPM_RH_OWNER and TPM_RH_LOCKOUT)
            default:
                hierarchy = TPM_RH_OWNER;
                break;
        }
        break;
    case TPM_HT_NV_INDEX:
        // hierarchy for NV index
        {
            NV_INDEX* nvIndex = NvGetIndexInfo(handle, NULL);
            pAssert(nvIndex != NULL);

            // If only the platform can delete the index, then it is
            // considered to be in the platform hierarchy, otherwise it
            // is in the owner hierarchy.
            if(IS_ATTRIBUTE(
                   nvIndex->publicArea.attributes, TPMA_NV, PLATFORMCREATE))
                hierarchy = TPM_RH_PLATFORM;
            else
                hierarchy = TPM_RH_OWNER;
        }
        break;
    case TPM_HT_TRANSIENT:
        // hierarchy for an object
        {
            OBJECT* object;
            object = HandleToObject(handle);
            if(object->attributes.ppsHierarchy)
            {
                hierarchy = TPM_RH_PLATFORM;
            }
            else if(object->attributes.epsHierarchy)
            {
                hierarchy = TPM_RH_ENDORSEMENT;
            }
            else if(object->attributes.spsHierarchy)
            {
                hierarchy = TPM_RH_OWNER;
            }
        }
        break;
    case TPM_HT_PCR:
        hierarchy = TPM_RH_OWNER;
        break;
    default:
        FAIL(FATAL_ERROR_INTERNAL);
        break;
}
```

```
    // this is unreachable but it provides a return value for the default
    // case which makes the complier happy
    return hierarchy;
}
```

7.184 /tpm/src/support/Global.c

```
//** Description
// This file will instance the TPM variables that are not stack allocated.

// Descriptions of global variables are in Global.h. There macro macro definitions
// that allows a variable to be instanced or simply defined as an external variable.
// When global.h is included from this .c file, GLOBAL_C is defined and values are
// instanced (and possibly initialized), but when global.h is included by any other
// file, they are simply defined as external values. DO NOT DEFINE GLOBAL_C IN ANY
// OTHER FILE.
//
// NOTE: This is a change from previous implementations where Global.h just contained
// the extern declaration and values were instanced in this file. This change keeps
// the definition and instance in one file making maintenance easier. The instanced
// data will still be in the global.obj file.
//
// The OIDs.h file works in a way that is similar to the Global.h with the definition
// of the values in OIDs.h such that they are instanced in global.obj. The macros
// that are defined in Global.h are used in OIDs.h in the same way as they are in
// Global.h.

//** Defines and Includes
#define GLOBAL_C
#include "Tpm.h"
#include "OIDs.h"

#if CC_CertifyX509
#  include "X509.h"
#endif  // CC_CertifyX509

// Global string constants for consistency in KDF function calls.
// These string constants are shared across functions to make sure that they
// are all using consistent string values.

// each instance must define a different struct since the buffer sizes vary.
#define TPM2B_STRING(name, value)                                           \
    typedef union name##_                                                   \
    {                                                                       \
        struct                                                              \
        {                                                                   \
            UINT16 size;                                                    \
            BYTE   buffer[sizeof(value)];                                   \
        } t;                                                                \
        TPM2B b;                                                            \
    } TPM2B_##name##_;                                                      \
    const TPM2B_##name##_ name##_data = {{sizeof(value), {value}}}; \
    const TPM2B*          name        = &name##_data.b

TPM2B_STRING(PRIMARY_OBJECT_CREATION, "Primary Object Creation");
TPM2B_STRING(CFB_KEY, "CFB");
TPM2B_STRING(CONTEXT_KEY, "CONTEXT");
TPM2B_STRING(INTEGRITY_KEY, "INTEGRITY");
TPM2B_STRING(SECRET_KEY, "SECRET");
TPM2B_STRING(HIERARCHY_PROOF_SECRET_LABEL, "H_PROOF_SECRET");
TPM2B_STRING(HIERARCHY_SEED_SECRET_LABEL, "H_SEED_SECRET");
TPM2B_STRING(HIERARCHY_FW_SECRET_LABEL, "H_FW_SECRET");
TPM2B_STRING(HIERARCHY_SVN_SECRET_LABEL, "H_SVN_SECRET");
TPM2B_STRING(SESSION_KEY, "ATH");
TPM2B_STRING(STORAGE_KEY, "STORAGE");
```

```
TPM2B_STRING(XOR_KEY, "XOR");
TPM2B_STRING(COMMIT_STRING, "ECDAA Commit");
TPM2B_STRING(DUPLICATE_STRING, "DUPLICATE");
TPM2B_STRING(IDENTITY_STRING, "IDENTITY");
TPM2B_STRING(OBFUSCATE_STRING, "OBFUSCATE");
#if ENABLE_SELF_TESTS
TPM2B_STRING(OAEP_TEST_STRING, "OAEP Test Value");
#endif  // ENABLE_SELF_TESTS

//*** g_rcIndex[]
const UINT16 g_rcIndex[15]  = {TPM_RC_1,
                               TPM_RC_2,
                               TPM_RC_3,
                               TPM_RC_4,
                               TPM_RC_5,
                               TPM_RC_6,
                               TPM_RC_7,
                               TPM_RC_8,
                               TPM_RC_9,
                               TPM_RC_A,
                               TPM_RC_B,
                               TPM_RC_C,
                               TPM_RC_D,
                               TPM_RC_E,
                               TPM_RC_F};

BOOL         g_manufactured = FALSE;
```

7.185 /tpm/src/support/Handle.c

```
//** Description
// This file contains the functions that return the type of a handle.

//** Includes
#include "Tpm.h"

//** Functions

//*** HandleGetType()
// This function returns the type of a handle which is the MSO of the handle.
TPM_HT
HandleGetType(TPM_HANDLE handle  // IN: a handle to be checked
)
{
    // return the upper bytes of input data
    return (TPM_HT)((handle & HR_RANGE_MASK) >> HR_SHIFT);
}

//*** NextPermanentHandle()
// This function returns the permanent handle that is equal to the input value or
// is the next higher value. If there is no handle with the input value and there
// is no next higher value, it returns 0:
TPM_HANDLE
NextPermanentHandle(TPM_HANDLE inHandle  // IN: the handle to check
)
{
    // If inHandle is below the start of the range of permanent handles
    // set it to the start and scan from there
    if(inHandle < TPM_RH_FIRST)
        inHandle = TPM_RH_FIRST;
    // scan from input value until we find an implemented permanent handle
    // or go out of range
    for(; inHandle <= TPM_RH_LAST; inHandle++)
    {
        // Skip over gaps in the reserved handle space.
```

```
        if(inHandle > TPM_RH_FW_NULL && inHandle < SVN_OWNER_FIRST)
            inHandle = SVN_OWNER_FIRST;
        if(inHandle > SVN_OWNER_FIRST && inHandle <= SVN_OWNER_LAST)
            inHandle = SVN_ENDORSEMENT_FIRST;
        if(inHandle > SVN_ENDORSEMENT_FIRST && inHandle <= SVN_ENDORSEMENT_LAST)
            inHandle = SVN_PLATFORM_FIRST;
        if(inHandle > SVN_PLATFORM_FIRST && inHandle <= SVN_PLATFORM_LAST)
            inHandle = SVN_NULL_FIRST;
        if(inHandle > SVN_NULL_FIRST)
            inHandle = TPM_RH_LAST;

        switch(inHandle)
        {
            case TPM_RH_OWNER:
            case TPM_RH_NULL:
            case TPM_RS_PW:
            case TPM_RH_LOCKOUT:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
            case TPM_RH_PLATFORM_NV:
#if FW_LIMITED_SUPPORT
            case TPM_RH_FW_OWNER:
            case TPM_RH_FW_ENDORSEMENT:
            case TPM_RH_FW_PLATFORM:
            case TPM_RH_FW_NULL:
#endif
#if SVN_LIMITED_SUPPORT
            case TPM_RH_SVN_OWNER_BASE:
            case TPM_RH_SVN_ENDORSEMENT_BASE:
            case TPM_RH_SVN_PLATFORM_BASE:
            case TPM_RH_SVN_NULL_BASE:
#endif
#if VENDOR_PERMANENT_AUTH_ENABLED == YES
            case VENDOR_PERMANENT_AUTH_HANDLE:
#endif
// Each of the implemented ACT
#define ACT_IMPLEMENTED_CASE(N) case TPM_RH_ACT_##N:

                FOR_EACH_ACT(ACT_IMPLEMENTED_CASE)

                return inHandle;
                break;
            default:
                break;
        }
    }
    // Out of range on the top
    return 0;
}

//*** PermanentCapGetHandles()
// This function returns a list of the permanent handles of PCR, started from
// 'handle'. If 'handle' is larger than the largest permanent handle, an empty list
// will be returned with 'more' set to NO.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
PermanentCapGetHandles(TPM_HANDLE   handle,     // IN: start handle
                       UINT32       count,      // IN: count of returned handles
                       TPML_HANDLE* handleList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;
```

```
    pAssert(HandleGetType(handle) == TPM_HT_PERMANENT);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES)
        count = MAX_CAP_HANDLES;

    // Iterate permanent handle range
    for(i = NextPermanentHandle(handle); i != 0; i = NextPermanentHandle(i + 1))
    {
        if(handleList->count < count)
        {
            // If we have not filled up the return list, add this permanent
            // handle to it
            handleList->handle[handleList->count] = i;
            handleList->count++;
        }
        else
        {
            // If the return list is full but we still have permanent handle
            // available, report this and stop iterating
            more = YES;
            break;
        }
    }
    return more;
}

//*** PermanentCapGetOneHandle()
// This function returns whether a permanent handle exists.
BOOL PermanentCapGetOneHandle(TPM_HANDLE handle)  // IN: handle
{
    UINT32 i;

    pAssert(HandleGetType(handle) == TPM_HT_PERMANENT);

    // Iterate permanent handle range
    for(i = NextPermanentHandle(handle); i != 0; i = NextPermanentHandle(i + 1))
    {
        if(i == handle)
        {
            return TRUE;
        }
    }
    return FALSE;
}

//*** PermanentHandleGetPolicy()
// This function returns a list of the permanent handles of PCR, started from
// 'handle'. If 'handle' is larger than the largest permanent handle, an empty list
// will be returned with 'more' set to NO.
//  Return Type: TPMI_YES_NO
//      YES         if there are more handles available
//      NO          all the available handles has been returned
TPMI_YES_NO
PermanentHandleGetPolicy(TPM_HANDLE handle,  // IN: start handle
                         UINT32     count,   // IN: max count of returned handles
                         TPML_TAGGED_POLICY* policyList  // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;

    pAssert(HandleGetType(handle) == TPM_HT_PERMANENT);
```

```
    // Initialize output handle list
    policyList->count = 0;

    // The maximum count of policies we may return is MAX_TAGGED_POLICIES
    if(count > MAX_TAGGED_POLICIES)
        count = MAX_TAGGED_POLICIES;

    // Iterate permanent handle range
    for(handle = NextPermanentHandle(handle); handle != 0;
        handle = NextPermanentHandle(handle + 1))
    {
        TPM2B_DIGEST policyDigest;
        TPM_ALG_ID   policyAlg;
        // Check to see if this permanent handle has a policy
        policyAlg = EntityGetAuthPolicy(handle, &policyDigest);
        if(policyAlg == TPM_ALG_ERROR)
            continue;
        if(policyList->count < count)
        {
            // If we have not filled up the return list, add this
            // policy to the list;
            policyList->policies[policyList->count].handle             = handle;
            policyList->policies[policyList->count].policyHash.hashAlg = policyAlg;
            MemoryCopy(&policyList->policies[policyList->count].policyHash.digest,
                       policyDigest.t.buffer,
                       policyDigest.t.size);
            policyList->count++;
        }
        else
        {
            // If the return list is full but we still have permanent handle
            // available, report this and stop iterating
            more = YES;
            break;
        }
    }
    return more;
}

//*** PermanentHandleGetOnePolicy()
// This function returns a permanent handle's policy, if present.
BOOL PermanentHandleGetOnePolicy(TPM_HANDLE          handle,  // IN: handle
                                 TPMS_TAGGED_POLICY* policy   // OUT: tagged policy
)
{
    pAssert(HandleGetType(handle) == TPM_HT_PERMANENT);

    if(NextPermanentHandle(handle) == handle)
    {
        TPM2B_DIGEST policyDigest;
        TPM_ALG_ID   policyAlg;
        // Check to see if this permanent handle has a policy
        policyAlg = EntityGetAuthPolicy(handle, &policyDigest);
        if(policyAlg == TPM_ALG_ERROR)
        {
            return FALSE;
        }
        policy->handle             = handle;
        policy->policyHash.hashAlg = policyAlg;
        MemoryCopy(
            &policy->policyHash.digest, policyDigest.t.buffer, policyDigest.t.size);
        return TRUE;
    }
    return FALSE;
}
```

7.186 /tpm/src/support/IoBuffers.c

```
//** Includes and Data Definitions

// This definition allows this module to "see" the values that are private
// to this module but kept in Global.c for ease of state migration.
#define IO_BUFFER_C
#include "Tpm.h"
#include "IoBuffers_fp.h"

//** Buffers and Functions

// These buffers are set aside to hold command and response values. In this
// implementation, it is not guaranteed that the code will stop accessing
// the s_actionInputBuffer before starting to put values in the
// s_actionOutputBuffer so different buffers are required.
//

//*** MemoryIoBufferAllocationReset()
// This function is used to reset the allocation of buffers.
void MemoryIoBufferAllocationReset(void)
{
    s_actionIoAllocation = 0;
}

//*** MemoryIoBufferZero()
// Function zeros the action I/O buffer at the end of a command. Calling this is
// not mandatory for proper functionality.
void MemoryIoBufferZero(void)
{
    memset(s_actionIoBuffer, 0, s_actionIoAllocation);
}

//*** MemoryGetInBuffer()
// This function returns the address of the buffer into which the
// command parameters will be unmarshaled in preparation for calling
// the command actions.
BYTE* MemoryGetInBuffer(UINT32 size  // Size, in bytes, required for the input
                                     // unmarshaling
)
{
    pAssert(size <= sizeof(s_actionIoBuffer));
// In this implementation, a static buffer is set aside for the command action
// buffers. The buffer is shared between input and output. This is because
// there is no need to allocate for the worst case input and worst case output
// at the same time.
// Round size up
#define UoM (sizeof(s_actionIoBuffer[0]))
    size = (size + (UoM - 1)) & (UINT32_MAX - (UoM - 1));
    memset(s_actionIoBuffer, 0, size);
    s_actionIoAllocation = size;
    return (BYTE*)&s_actionIoBuffer[0];
}

//*** MemoryGetOutBuffer()
// This function returns the address of the buffer into which the command
// action code places its output values.
BYTE* MemoryGetOutBuffer(UINT32 size  // required size of the buffer
)
{
    BYTE* retVal = (BYTE*)(&s_actionIoBuffer[s_actionIoAllocation / UoM]);
    pAssert((size + s_actionIoAllocation) < (sizeof(s_actionIoBuffer)));
    // In this implementation, a static buffer is set aside for the command action
    // output buffer.
    memset(retVal, 0, size);
    s_actionIoAllocation += size;
```

```
    return retVal;
}

//*** IsLabelProperlyFormatted()
// This function checks that a label is a null-terminated string.
// NOTE: this function is here because there was no better place for it.
//  Return Type: BOOL
//      TRUE(1)         string is null terminated
//      FALSE(0)        string is not null terminated
BOOL IsLabelProperlyFormatted(TPM2B* x)
{
    return (((x)->size == 0) || ((x)->buffer[(x)->size - 1] == 0));
}
```

7.187 /tpm/src/support/Locality.c

```
//** Includes
#include "Tpm.h"

//** LocalityGetAttributes()
// This function will convert a locality expressed as an integer into
// TPMA_LOCALITY form.
//
// The function returns the locality attribute.
TPMA_LOCALITY
LocalityGetAttributes(UINT8 locality  // IN: locality value
)
{
    TPMA_LOCALITY locality_attributes;
    BYTE*         localityAsByte = (BYTE*)&locality_attributes;

    MemorySet(&locality_attributes, 0, sizeof(TPMA_LOCALITY));
    switch(locality)
    {
        case 0:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_ZERO);
            break;
        case 1:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_ONE);
            break;
        case 2:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_TWO);
            break;
        case 3:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_THREE);
            break;
        case 4:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_FOUR);
            break;
        default:
            pAssert(locality > 31);
            *localityAsByte = locality;
            break;
    }
    return locality_attributes;
}
```

7.188 /tpm/src/support/Manufacture.c

```
//** Description
// This file contains the function that performs the "manufacturing" of the TPM
// in a simulated environment. These functions should not be used outside of
// a manufacturing or simulation environment.
```

```
//** Includes and Data Definitions
#define MANUFACTURE_C
#include "Tpm.h"
#include "TpmSizeChecks_fp.h"

//** Functions

//*** TPM_Manufacture()
// This function initializes the TPM values in preparation for the TPM's first
// use. This function will fail if previously called. The TPM can be re-manufactured
// by calling TPM_Teardown() first and then calling this function again.
// NV must be enabled first (typically with NvPowerOn() via _TPM_Init)
//
// return type: int
//      -2          NV System not available
//      -1          FAILURE - System is incorrectly compiled.
//      0           success
//      1           manufacturing process previously performed
LIB_EXPORT int TPM_Manufacture(
    int firstTime  // IN: indicates if this is the first call from
                   //     main()
)
{
    TPM_SU orderlyShutdown;

#if RUNTIME_SIZE_CHECKS
    // Call the function to verify the sizes of values that result from different
    // compile options.
    if(!TpmSizeChecks())
        return MANUF_INVALID_CONFIG;
#endif
#if LIBRARY_COMPATIBILITY_CHECK
    // Make sure that the attached library performs as expected.
    if(!ExtMath_Debug_CompatibilityCheck())
        return MANUF_INVALID_CONFIG;
#endif

    // If TPM has been manufactured, return indication.
    if(!firstTime && g_manufactured)
        return MANUF_ALREADY_DONE;

    // trigger failure mode if called in error.
    int nvReadyState = _plat__GetNvReadyState();
    pAssert(nvReadyState == NV_READY);  // else failure mode
    if(nvReadyState != NV_READY)
    {
        return MANUF_NV_NOT_READY;
    }

    // Do power on initializations of the cryptographic libraries.
    CryptInit();

    s_DAPendingOnNV = FALSE;

    // initialize NV
    NvManufacture();

    // Clear the magic value in the DRBG state
    go.drbgState.magic = 0;

    CryptStartup(SU_RESET);

    // default configuration for PCR
    PCRManufacture();

    // initialize pre-installed hierarchy data
```

```
    // This should happen after NV is initialized because hierarchy data is
    // stored in NV.
    HierarchyPreInstall_Init();

    // initialize dictionary attack parameters
    DAPreInstall_Init();

    // initialize PP list
    PhysicalPresencePreInstall_Init();

    // initialize command audit list
    CommandAuditPreInstall_Init();

    // first start up is required to be Startup(CLEAR)
    orderlyShutdown = TPM_SU_CLEAR;
    NV_WRITE_PERSISTENT(orderlyState, orderlyShutdown);

    // initialize the firmware version
    gp.firmwareV1 = _plat__GetTpmFirmwareVersionHigh();
    gp.firmwareV2 = _plat__GetTpmFirmwareVersionLow();

    _plat__GetPlatformManufactureData(gp.platformReserved,
                                      sizeof(gp.platformReserved));
    NV_SYNC_PERSISTENT(platformReserved);

    NV_SYNC_PERSISTENT(firmwareV1);
    NV_SYNC_PERSISTENT(firmwareV2);

    // initialize the total reset counter to 0
    gp.totalResetCount = 0;
    NV_SYNC_PERSISTENT(totalResetCount);

    // initialize the clock stuff
    go.clock     = 0;
    go.clockSafe = YES;

    NvWrite(NV_ORDERLY_DATA, sizeof(ORDERLY_DATA), &go);

    // Commit NV writes.  Manufacture process is an artificial process existing
    // only in simulator environment and it is not defined in the specification
    // that what should be the expected behavior if the NV write fails at this
    // point.  Therefore, it is assumed the NV write here is always success and
    // no return code of this function is checked.
    NvCommit();

    g_manufactured = TRUE;

    return MANUF_OK;
}

//*** TPM_TearDown()
// This function prepares the TPM for re-manufacture. It should not be implemented
// in anything other than a simulated TPM.
//
// In this implementation, all that is needs is to stop the cryptographic units
// and set a flag to indicate that the TPM can be re-manufactured. This should
// be all that is necessary to start the manufacturing process again.
//  Return Type: int
//      0        success
//      1        TPM not previously manufactured
LIB_EXPORT int TPM_TearDown(void)
{
    g_manufactured = FALSE;
    _plat__TearDown();
    return TEARDOWN_OK;
}
```

```
//*** TpmEndSimulation()
// This function is called at the end of the simulation run. It is used to provoke
// printing of any statistics that might be needed.
LIB_EXPORT void TpmEndSimulation(void)
{
#if SIMULATION
    HashLibSimulationEnd();
    SymLibSimulationEnd();
    MathLibSimulationEnd();
#  if ALG_RSA
    RsaSimulationEnd();
#  endif
#  if ALG_ECC
    EccSimulationEnd();
#  endif
#endif  // SIMULATION
}
```

7.189 /tpm/src/support/Marshal.c

```
// FILE GENERATED BY TpmExtractCode: DO NOT EDIT

#include "Tpm.h"
#if !TABLE_DRIVEN_MARSHAL
#  include "Marshal_fp.h"

// Table "Definition of Base Types" (Part 2: Structures)
//   UINT8 definition
TPM_RC
UINT8_Unmarshal(UINT8* target, BYTE** buffer, INT32* size)
{
    if((*size -= 1) < 0)
        return TPM_RC_INSUFFICIENT;
    *target = BYTE_ARRAY_TO_UINT8(*buffer);
    *buffer += 1;
    return TPM_RC_SUCCESS;
}
UINT16
UINT8_Marshal(UINT8* source, BYTE** buffer, INT32* size)
{
    if(buffer != 0)
    {
        if((size == 0) || ((*size -= 1) >= 0))
        {
            UINT8_TO_BYTE_ARRAY(*source, *buffer);
            *buffer += 1;
        }
        pAssert(size == 0 || (*size >= 0));
    }
    return (1);
}

//   BYTE definition
#  if !USE_MARSHALING_DEFINES
TPM_RC
BYTE_Unmarshal(BYTE* target, BYTE** buffer, INT32* size)
{
    return UINT8_Unmarshal((UINT8*)target, buffer, size);
}
UINT16
BYTE_Marshal(BYTE* source, BYTE** buffer, INT32* size)
{
    return UINT8_Marshal((UINT8*)source, buffer, size);
}
```

```
#  endif  // !USE_MARSHALING_DEFINES

//   INT8 definition
#  if !USE_MARSHALING_DEFINES
TPM_RC
INT8_Unmarshal(INT8* target, BYTE** buffer, INT32* size)
{
    return UINT8_Unmarshal((UINT8*)target, buffer, size);
}
UINT16
INT8_Marshal(INT8* source, BYTE** buffer, INT32* size)
{
    return UINT8_Marshal((UINT8*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

//   UINT16 definition
TPM_RC
UINT16_Unmarshal(UINT16* target, BYTE** buffer, INT32* size)
{
    if((*size -= 2) < 0)
        return TPM_RC_INSUFFICIENT;
    *target = BYTE_ARRAY_TO_UINT16(*buffer);
    *buffer += 2;
    return TPM_RC_SUCCESS;
}
UINT16
UINT16_Marshal(UINT16* source, BYTE** buffer, INT32* size)
{
    if(buffer != 0)
    {
        if((size == 0) || ((*size -= 2) >= 0))
        {
            UINT16_TO_BYTE_ARRAY(*source, *buffer);
            *buffer += 2;
        }
        pAssert(size == 0 || (*size >= 0));
    }
    return (2);
}

//   INT16 definition
#  if !USE_MARSHALING_DEFINES
TPM_RC
INT16_Unmarshal(INT16* target, BYTE** buffer, INT32* size)
{
    return UINT16_Unmarshal((UINT16*)target, buffer, size);
}
UINT16
INT16_Marshal(INT16* source, BYTE** buffer, INT32* size)
{
    return UINT16_Marshal((UINT16*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

//   UINT32 definition
TPM_RC
UINT32_Unmarshal(UINT32* target, BYTE** buffer, INT32* size)
{
    if((*size -= 4) < 0)
        return TPM_RC_INSUFFICIENT;
    *target = BYTE_ARRAY_TO_UINT32(*buffer);
    *buffer += 4;
    return TPM_RC_SUCCESS;
}
UINT16
```

```
UINT32_Marshal(UINT32* source, BYTE** buffer, INT32* size)
{
    if(buffer != 0)
    {
        if((size == 0) || ((*size -= 4) >= 0))
        {
            UINT32_TO_BYTE_ARRAY(*source, *buffer);
            *buffer += 4;
        }
        pAssert(size == 0 || (*size >= 0));
    }
    return (4);
}

//   INT32 definition
#  if !USE_MARSHALING_DEFINES
TPM_RC
INT32_Unmarshal(INT32* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
INT32_Marshal(INT32* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

//   UINT64 definition
TPM_RC
UINT64_Unmarshal(UINT64* target, BYTE** buffer, INT32* size)
{
    if((*size -= 8) < 0)
        return TPM_RC_INSUFFICIENT;
    *target = BYTE_ARRAY_TO_UINT64(*buffer);
    *buffer += 8;
    return TPM_RC_SUCCESS;
}
UINT16
UINT64_Marshal(UINT64* source, BYTE** buffer, INT32* size)
{
    if(buffer != 0)
    {
        if((size == 0) || ((*size -= 8) >= 0))
        {
            UINT64_TO_BYTE_ARRAY(*source, *buffer);
            *buffer += 8;
        }
        pAssert(size == 0 || (*size >= 0));
    }
    return (8);
}

//   INT64 definition
#  if !USE_MARSHALING_DEFINES
TPM_RC
INT64_Unmarshal(INT64* target, BYTE** buffer, INT32* size)
{
    return UINT64_Unmarshal((UINT64*)target, buffer, size);
}
UINT16
INT64_Marshal(INT64* source, BYTE** buffer, INT32* size)
{
    return UINT64_Marshal((UINT64*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of Types for Documentation Clarity" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_ALGORITHM_ID_Unmarshal(TPM_ALGORITHM_ID* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_ALGORITHM_ID_Marshal(TPM_ALGORITHM_ID* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
TPM_RC
TPM_AUTHORIZATION_SIZE_Unmarshal(
    TPM_AUTHORIZATION_SIZE* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_AUTHORIZATION_SIZE_Marshal(
    TPM_AUTHORIZATION_SIZE* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
TPM_RC
TPM_KEY_BITS_Unmarshal(TPM_KEY_BITS* target, BYTE** buffer, INT32* size)
{
    return UINT16_Unmarshal((UINT16*)target, buffer, size);
}
UINT16
TPM_KEY_BITS_Marshal(TPM_KEY_BITS* source, BYTE** buffer, INT32* size)
{
    return UINT16_Marshal((UINT16*)source, buffer, size);
}
TPM_RC
TPM_KEY_SIZE_Unmarshal(TPM_KEY_SIZE* target, BYTE** buffer, INT32* size)
{
    return UINT16_Unmarshal((UINT16*)target, buffer, size);
}
UINT16
TPM_KEY_SIZE_Marshal(TPM_KEY_SIZE* source, BYTE** buffer, INT32* size)
{
    return UINT16_Marshal((UINT16*)source, buffer, size);
}
TPM_RC
TPM_MODIFIER_INDICATOR_Unmarshal(
    TPM_MODIFIER_INDICATOR* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_MODIFIER_INDICATOR_Marshal(
    TPM_MODIFIER_INDICATOR* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
TPM_RC
TPM_PARAMETER_SIZE_Unmarshal(TPM_PARAMETER_SIZE* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_PARAMETER_SIZE_Marshal(TPM_PARAMETER_SIZE* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
```

```
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_CONSTANTS32 Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_CONSTANTS32_Marshal(TPM_CONSTANTS32* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_ALG_ID Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_ALG_ID_Unmarshal(TPM_ALG_ID* target, BYTE** buffer, INT32* size)
{
    return UINT16_Unmarshal((UINT16*)target, buffer, size);
}
UINT16
TPM_ALG_ID_Marshal(TPM_ALG_ID* source, BYTE** buffer, INT32* size)
{
    return UINT16_Marshal((UINT16*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_ECC_CURVE Constants" (Part 2: Structures)
TPM_RC
TPM_ECC_CURVE_Unmarshal(TPM_ECC_CURVE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ECC_NIST_P192
            case TPM_ECC_NIST_P192:
#  endif  // ECC_NIST_P192
#  if ECC_NIST_P224
            case TPM_ECC_NIST_P224:
#  endif  // ECC_NIST_P224
#  if ECC_NIST_P256
            case TPM_ECC_NIST_P256:
#  endif  // ECC_NIST_P256
#  if ECC_NIST_P384
            case TPM_ECC_NIST_P384:
#  endif  // ECC_NIST_P384
#  if ECC_NIST_P521
            case TPM_ECC_NIST_P521:
#  endif  // ECC_NIST_P521
#  if ECC_BN_P256
            case TPM_ECC_BN_P256:
#  endif  // ECC_BN_P256
#  if ECC_BN_P638
            case TPM_ECC_BN_P638:
#  endif  // ECC_BN_P638
#  if ECC_SM2_P256
            case TPM_ECC_SM2_P256:
#  endif  // ECC_SM2_P256
#  if ECC_BP_P256_R1
            case TPM_ECC_BP_P256_R1:
#  endif  // ECC_BP_P256_R1
#  if ECC_BP_P384_R1
            case TPM_ECC_BP_P384_R1:
#  endif  // ECC_BP_P384_R1
```

```
#  if ECC_BP_P512_R1
            case TPM_ECC_BP_P512_R1:
#  endif  // ECC_BP_P512_R1
#  if ECC_CURVE_25519
            case TPM_ECC_CURVE_25519:
#  endif  // ECC_CURVE_25519
#  if ECC_CURVE_448
            case TPM_ECC_CURVE_448:
#  endif  // ECC_CURVE_448
                break;
            default:
                result = TPM_RC_CURVE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_ECC_CURVE_Marshal(TPM_ECC_CURVE* source, BYTE** buffer, INT32* size)
{
    return UINT16_Marshal((UINT16*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_CC Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_CC_Unmarshal(TPM_CC* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_CC_Marshal(TPM_CC* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_RC Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_RC_Marshal(TPM_RC* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_CLOCK_ADJUST Constants" (Part 2: Structures)
TPM_RC
TPM_CLOCK_ADJUST_Unmarshal(TPM_CLOCK_ADJUST* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = INT8_Unmarshal((INT8*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_CLOCK_COARSE_SLOWER:
            case TPM_CLOCK_MEDIUM_SLOWER:
            case TPM_CLOCK_FINE_SLOWER:
            case TPM_CLOCK_NO_CHANGE:
            case TPM_CLOCK_FINE_FASTER:
            case TPM_CLOCK_MEDIUM_FASTER:
            case TPM_CLOCK_COARSE_FASTER:
                break;
```

```
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPM_EO Constants" (Part 2: Structures)
TPM_RC
TPM_EO_Unmarshal(TPM_EO* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_EO_EQ:
            case TPM_EO_NEQ:
            case TPM_EO_SIGNED_GT:
            case TPM_EO_UNSIGNED_GT:
            case TPM_EO_SIGNED_LT:
            case TPM_EO_UNSIGNED_LT:
            case TPM_EO_SIGNED_GE:
            case TPM_EO_UNSIGNED_GE:
            case TPM_EO_SIGNED_LE:
            case TPM_EO_UNSIGNED_LE:
            case TPM_EO_BITSET:
            case TPM_EO_BITCLEAR:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_EO_Marshal(TPM_EO* source, BYTE** buffer, INT32* size)
{
    return UINT16_Marshal((UINT16*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_ST Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_ST_Unmarshal(TPM_ST* target, BYTE** buffer, INT32* size)
{
    return UINT16_Unmarshal((UINT16*)target, buffer, size);
}
UINT16
TPM_ST_Marshal(TPM_ST* source, BYTE** buffer, INT32* size)
{
    return UINT16_Marshal((UINT16*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_SU Constants" (Part 2: Structures)
TPM_RC
TPM_SU_Unmarshal(TPM_SU* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)target, buffer, size);
```

```
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_SU_CLEAR:
            case TPM_SU_STATE:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPM_SE Constants" (Part 2: Structures)
TPM_RC
TPM_SE_Unmarshal(TPM_SE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT8_Unmarshal((UINT8*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_SE_HMAC:
            case TPM_SE_POLICY:
            case TPM_SE_TRIAL:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPM_CAP Constants" (Part 2: Structures)
TPM_RC
TPM_CAP_Unmarshal(TPM_CAP* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_CAP_ALGS:
            case TPM_CAP_HANDLES:
            case TPM_CAP_COMMANDS:
            case TPM_CAP_PP_COMMANDS:
            case TPM_CAP_AUDIT_COMMANDS:
            case TPM_CAP_PCRS:
            case TPM_CAP_TPM_PROPERTIES:
            case TPM_CAP_PCR_PROPERTIES:
#  if ALG_ECC
            case TPM_CAP_ECC_CURVES:
#  endif  // ALG_ECC
            case TPM_CAP_AUTH_POLICIES:
            case TPM_CAP_ACT:
            case TPM_CAP_VENDOR_PROPERTY:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
```

```
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_CAP_Marshal(TPM_CAP* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_PT Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_PT_Unmarshal(TPM_PT* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_PT_Marshal(TPM_PT* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_PT_PCR Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_PT_PCR_Unmarshal(TPM_PT_PCR* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_PT_PCR_Marshal(TPM_PT_PCR* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_PS Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_PS_Marshal(TPM_PS* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for Handles" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_HANDLE_Unmarshal(TPM_HANDLE* target, BYTE** buffer, INT32* size)
{
    return UINT32_Unmarshal((UINT32*)target, buffer, size);
}
UINT16
TPM_HANDLE_Marshal(TPM_HANDLE* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_HT Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_HT_Unmarshal(TPM_HT* target, BYTE** buffer, INT32* size)
```

```
{
    return UINT8_Unmarshal((UINT8*)target, buffer, size);
}
UINT16
TPM_HT_Marshal(TPM_HT* source, BYTE** buffer, INT32* size)
{
    return UINT8_Marshal((UINT8*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_RH Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_RH_Unmarshal(TPM_RH* target, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
}
UINT16
TPM_RH_Marshal(TPM_RH* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_HC Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM_HC_Unmarshal(TPM_HC* target, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
}
UINT16
TPM_HC_Marshal(TPM_HC* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_ALGORITHM Bits" (Part 2: Structures)
TPM_RC
TPMA_ALGORITHM_Unmarshal(TPMA_ALGORITHM* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // check that no reserved bits are set
        if(*((UINT32*)target) & (UINT32)0xfffff8f0)
            result = TPM_RC_RESERVED_BITS;
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_ALGORITHM_Marshal(TPMA_ALGORITHM* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_OBJECT Bits" (Part 2: Structures)
TPM_RC
TPMA_OBJECT_Unmarshal(TPMA_OBJECT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)target, buffer, size);
```

```
    if(result == TPM_RC_SUCCESS)
    {
        // check that no reserved bits are set
        if(*((UINT32*)target) & (UINT32)0xfff0f001)
            result = TPM_RC_RESERVED_BITS;
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_OBJECT_Marshal(TPMA_OBJECT* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_SESSION Bits" (Part 2: Structures)
TPM_RC
TPMA_SESSION_Unmarshal(TPMA_SESSION* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT8_Unmarshal((UINT8*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // check that no reserved bits are set
        if(*((UINT8*)target) & (UINT8)0x18)
            result = TPM_RC_RESERVED_BITS;
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_SESSION_Marshal(TPMA_SESSION* source, BYTE** buffer, INT32* size)
{
    return UINT8_Marshal((UINT8*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_LOCALITY Bits" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMA_LOCALITY_Unmarshal(TPMA_LOCALITY* target, BYTE** buffer, INT32* size)
{
    return UINT8_Unmarshal((UINT8*)target, buffer, size);
}
UINT16
TPMA_LOCALITY_Marshal(TPMA_LOCALITY* source, BYTE** buffer, INT32* size)
{
    return UINT8_Marshal((UINT8*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_PERMANENT Bits" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_PERMANENT_Marshal(TPMA_PERMANENT* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_STARTUP_CLEAR Bits" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_STARTUP_CLEAR_Marshal(TPMA_STARTUP_CLEAR* source, BYTE** buffer, INT32* size)
{
```

```
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_MEMORY Bits" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_MEMORY_Marshal(TPMA_MEMORY* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_CC Bits" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_CC_Marshal(TPMA_CC* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_MODES Bits" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_MODES_Marshal(TPMA_MODES* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_ACT Bits" (Part 2: Structures)
TPM_RC
TPMA_ACT_Unmarshal(TPMA_ACT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // check that no reserved bits are set
        if(*((UINT32*)target) & (UINT32)0xfffffffc)
            result = TPM_RC_RESERVED_BITS;
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_ACT_Marshal(TPMA_ACT* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_YES_NO Type" (Part 2: Structures)
TPM_RC
TPMI_YES_NO_Unmarshal(TPMI_YES_NO* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = BYTE_Unmarshal((BYTE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case NO:
            case YES:
                break;
```

```
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_YES_NO_Marshal(TPMI_YES_NO* source, BYTE** buffer, INT32* size)
{
    return BYTE_Marshal((BYTE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_OBJECT Type" (Part 2: Structures)
TPM_RC
TPMI_DH_OBJECT_Unmarshal(
    TPMI_DH_OBJECT* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS) && ((*target != TPM_RH_NULL) || !flag)
       && ((*target < TRANSIENT_FIRST) || (*target > TRANSIENT_LAST))
       && ((*target < PERSISTENT_FIRST) || (*target > PERSISTENT_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_OBJECT_Marshal(TPMI_DH_OBJECT* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_PARENT Type" (Part 2: Structures)
TPM_RC
TPMI_DH_PARENT_Unmarshal(TPMI_DH_PARENT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
            case TPM_RH_FW_OWNER:
            case TPM_RH_FW_ENDORSEMENT:
            case TPM_RH_FW_PLATFORM:
                break;
            default:
                if((*target != TPM_RH_NULL) && (*target != TPM_RH_FW_NULL)
                   && ((*target < SVN_NULL_FIRST) || (*target > SVN_NULL_LAST))
                   && ((*target < SVN_OWNER_FIRST) || (*target > SVN_OWNER_LAST))
                   && ((*target < SVN_ENDORSEMENT_FIRST)
                       || (*target > SVN_ENDORSEMENT_LAST))
                   && ((*target < SVN_PLATFORM_FIRST)
                       || (*target > SVN_PLATFORM_LAST))
                   && ((*target < TRANSIENT_FIRST) || (*target > TRANSIENT_LAST))
                   && ((*target < PERSISTENT_FIRST) || (*target > PERSISTENT_LAST)))
                    result = TPM_RC_VALUE;
                break;
        }
```

```
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_PARENT_Marshal(TPMI_DH_PARENT* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_PERSISTENT Type" (Part 2: Structures)
TPM_RC
TPMI_DH_PERSISTENT_Unmarshal(TPMI_DH_PERSISTENT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < PERSISTENT_FIRST) || (*target > PERSISTENT_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_PERSISTENT_Marshal(TPMI_DH_PERSISTENT* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_ENTITY Type" (Part 2: Structures)
TPM_RC
TPMI_DH_ENTITY_Unmarshal(
    TPMI_DH_ENTITY* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
            case TPM_RH_LOCKOUT:
                break;
            default:
                if(((*target != TPM_RH_NULL) || !flag)
                   && ((*target < TRANSIENT_FIRST) || (*target > TRANSIENT_LAST))
                   && ((*target < PERSISTENT_FIRST) || (*target > PERSISTENT_LAST))
                   && ((*target < NV_INDEX_FIRST) || (*target > NV_INDEX_LAST))
                   && (*target > PCR_LAST)
                   && ((*target < TPM_RH_AUTH_00) || (*target > TPM_RH_AUTH_FF)))
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_DH_PCR Type" (Part 2: Structures)
TPM_RC
TPMI_DH_PCR_Unmarshal(TPMI_DH_PCR* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
```

```
    if((result == TPM_RC_SUCCESS) && ((*target != TPM_RH_NULL) || !flag)
       && (*target > PCR_LAST))
        result = TPM_RC_VALUE;
    return result;
}

// Table "Definition of TPMI_SH_AUTH_SESSION Type" (Part 2: Structures)
TPM_RC
TPMI_SH_AUTH_SESSION_Unmarshal(
    TPMI_SH_AUTH_SESSION* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS) && ((*target != TPM_RS_PW) || !flag)
       && ((*target < HMAC_SESSION_FIRST) || (*target > HMAC_SESSION_LAST))
       && ((*target < POLICY_SESSION_FIRST) || (*target > POLICY_SESSION_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_SH_AUTH_SESSION_Marshal(TPMI_SH_AUTH_SESSION* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_SH_HMAC Type" (Part 2: Structures)
TPM_RC
TPMI_SH_HMAC_Unmarshal(TPMI_SH_HMAC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < HMAC_SESSION_FIRST) || (*target > HMAC_SESSION_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_SH_HMAC_Marshal(TPMI_SH_HMAC* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_SH_POLICY Type" (Part 2: Structures)
TPM_RC
TPMI_SH_POLICY_Unmarshal(TPMI_SH_POLICY* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < POLICY_SESSION_FIRST) || (*target > POLICY_SESSION_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_SH_POLICY_Marshal(TPMI_SH_POLICY* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_CONTEXT Type" (Part 2: Structures)
```

```
TPM_RC
TPMI_DH_CONTEXT_Unmarshal(TPMI_DH_CONTEXT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < HMAC_SESSION_FIRST) || (*target > HMAC_SESSION_LAST))
       && ((*target < POLICY_SESSION_FIRST) || (*target > POLICY_SESSION_LAST))
       && ((*target < TRANSIENT_FIRST) || (*target > TRANSIENT_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_CONTEXT_Marshal(TPMI_DH_CONTEXT* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_DH_SAVED Type" (Part 2: Structures)
TPM_RC
TPMI_DH_SAVED_Unmarshal(TPMI_DH_SAVED* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case 0x80000000:
            case 0x80000001:
            case 0x80000002:
                break;
            default:
                if(((*target < HMAC_SESSION_FIRST) || (*target > HMAC_SESSION_LAST))
                   && ((*target < POLICY_SESSION_FIRST)
                       || (*target > POLICY_SESSION_LAST)))
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_DH_SAVED_Marshal(TPMI_DH_SAVED* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_HIERARCHY Type" (Part 2: Structures)
TPM_RC
TPMI_RH_HIERARCHY_Unmarshal(TPMI_RH_HIERARCHY* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
            case TPM_RH_FW_OWNER:
```

```
            case TPM_RH_FW_ENDORSEMENT:
            case TPM_RH_FW_PLATFORM:
                break;
            default:
                if((*target != TPM_RH_NULL) && (*target != TPM_RH_FW_NULL)
                   && ((*target < SVN_NULL_FIRST) || (*target > SVN_NULL_LAST))
                   && ((*target < SVN_OWNER_FIRST) || (*target > SVN_OWNER_LAST))
                   && ((*target < SVN_ENDORSEMENT_FIRST)
                       || (*target > SVN_ENDORSEMENT_LAST))
                   && ((*target < SVN_PLATFORM_FIRST)
                       || (*target > SVN_PLATFORM_LAST)))
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_HIERARCHY_Marshal(TPMI_RH_HIERARCHY* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_ENABLES Type" (Part 2: Structures)
TPM_RC
TPMI_RH_ENABLES_Unmarshal(
    TPMI_RH_ENABLES* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
            case TPM_RH_PLATFORM_NV:
                break;
            default:
                if((*target != TPM_RH_NULL) || !flag)
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_ENABLES_Marshal(TPMI_RH_ENABLES* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_HIERARCHY_AUTH Type" (Part 2: Structures)
TPM_RC
TPMI_RH_HIERARCHY_AUTH_Unmarshal(
    TPMI_RH_HIERARCHY_AUTH* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
```

```
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
            case TPM_RH_LOCKOUT:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_RH_HIERARCHY_POLICY Type" (Part 2: Structures)
TPM_RC
TPMI_RH_HIERARCHY_POLICY_Unmarshal(
    TPMI_RH_HIERARCHY_POLICY* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
            case TPM_RH_LOCKOUT:
                break;
            default:
                if((*target < TPM_RH_ACT_0) || (*target > TPM_RH_ACT_F))
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_RH_BASE_HIERARCHY Type" (Part 2: Structures)
TPM_RC
TPMI_RH_BASE_HIERARCHY_Unmarshal(
    TPMI_RH_BASE_HIERARCHY* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_BASE_HIERARCHY_Marshal(
    TPMI_RH_BASE_HIERARCHY* source, BYTE** buffer, INT32* size)
```

```
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_PLATFORM Type" (Part 2: Structures)
TPM_RC
TPMI_RH_PLATFORM_Unmarshal(TPMI_RH_PLATFORM* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_PLATFORM:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_RH_OWNER Type" (Part 2: Structures)
TPM_RC
TPMI_RH_OWNER_Unmarshal(TPMI_RH_OWNER* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
                break;
            default:
                if((*target != TPM_RH_NULL) || !flag)
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_RH_ENDORSEMENT Type" (Part 2: Structures)
TPM_RC
TPMI_RH_ENDORSEMENT_Unmarshal(
    TPMI_RH_ENDORSEMENT* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_ENDORSEMENT:
                break;
            default:
                if((*target != TPM_RH_NULL) || !flag)
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
```

```
}

// Table "Definition of TPMI_RH_PROVISION Type" (Part 2: Structures)
TPM_RC
TPMI_RH_PROVISION_Unmarshal(TPMI_RH_PROVISION* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_PLATFORM:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_RH_CLEAR Type" (Part 2: Structures)
TPM_RC
TPMI_RH_CLEAR_Unmarshal(TPMI_RH_CLEAR* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_PLATFORM:
            case TPM_RH_LOCKOUT:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_RH_NV_AUTH Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_AUTH_Unmarshal(TPMI_RH_NV_AUTH* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_OWNER:
            case TPM_RH_PLATFORM:
                break;
            default:
                if((*target < NV_INDEX_FIRST) || (*target > NV_INDEX_LAST))
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
```

```
// Table "Definition of TPMI_RH_LOCKOUT Type" (Part 2: Structures)
TPM_RC
TPMI_RH_LOCKOUT_Unmarshal(TPMI_RH_LOCKOUT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_RH_LOCKOUT:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}

// Table "Definition of TPMI_RH_NV_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_INDEX_Unmarshal(TPMI_RH_NV_INDEX* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < NV_INDEX_FIRST) || (*target > NV_INDEX_LAST))
       && ((*target < EXTERNAL_NV_FIRST) || (*target > EXTERNAL_NV_LAST))
       && ((*target < PERMANENT_NV_FIRST) || (*target > PERMANENT_NV_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_NV_INDEX_Marshal(TPMI_RH_NV_INDEX* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_NV_DEFINED_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_DEFINED_INDEX_Unmarshal(
    TPMI_RH_NV_DEFINED_INDEX* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < NV_INDEX_FIRST) || (*target > NV_INDEX_LAST))
       && ((*target < EXTERNAL_NV_FIRST) || (*target > EXTERNAL_NV_LAST)))
        result = TPM_RC_VALUE;
    return result;
}

// Table "Definition of TPMI_RH_NV_LEGACY_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_LEGACY_INDEX_Unmarshal(
    TPMI_RH_NV_LEGACY_INDEX* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < NV_INDEX_FIRST) || (*target > NV_INDEX_LAST)))
        result = TPM_RC_VALUE;
    return result;
```

```
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_NV_LEGACY_INDEX_Marshal(
    TPMI_RH_NV_LEGACY_INDEX* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_NV_EXP_INDEX Type" (Part 2: Structures)
TPM_RC
TPMI_RH_NV_EXP_INDEX_Unmarshal(
    TPMI_RH_NV_EXP_INDEX* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < EXTERNAL_NV_FIRST) || (*target > EXTERNAL_NV_LAST)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_NV_EXP_INDEX_Marshal(TPMI_RH_NV_EXP_INDEX* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_RH_AC Type" (Part 2: Structures)
TPM_RC
TPMI_RH_AC_Unmarshal(TPMI_RH_AC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS) && ((*target < AC_FIRST) || (*target > AC_LAST)))
        result = TPM_RC_VALUE;
    return result;
}

// Table "Definition of TPMI_RH_ACT Type" (Part 2: Structures)
TPM_RC
TPMI_RH_ACT_Unmarshal(TPMI_RH_ACT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HANDLE_Unmarshal((TPM_HANDLE*)target, buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((*target < TPM_RH_ACT_0) || (*target > TPM_RH_ACT_F)))
        result = TPM_RC_VALUE;
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_RH_ACT_Marshal(TPMI_RH_ACT* source, BYTE** buffer, INT32* size)
{
    return TPM_HANDLE_Marshal((TPM_HANDLE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_HASH Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_HASH_Unmarshal(TPMI_ALG_HASH* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
```

```
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#   if ALG_SHA1
            case TPM_ALG_SHA1:
#   endif  // ALG_SHA1
#   if ALG_SHA256
            case TPM_ALG_SHA256:
#   endif  // ALG_SHA256
#   if ALG_SHA384
            case TPM_ALG_SHA384:
#   endif  // ALG_SHA384
#   if ALG_SHA512
            case TPM_ALG_SHA512:
#   endif  // ALG_SHA512
#   if ALG_SHA256_192
            case TPM_ALG_SHA256_192:
#   endif  // ALG_SHA256_192
#   if ALG_SM3_256
            case TPM_ALG_SM3_256:
#   endif  // ALG_SM3_256
#   if ALG_SHA3_256
            case TPM_ALG_SHA3_256:
#   endif  // ALG_SHA3_256
#   if ALG_SHA3_384
            case TPM_ALG_SHA3_384:
#   endif  // ALG_SHA3_384
#   if ALG_SHA3_512
            case TPM_ALG_SHA3_512:
#   endif  // ALG_SHA3_512
#   if ALG_SHAKE256_192
            case TPM_ALG_SHAKE256_192:
#   endif  // ALG_SHAKE256_192
#   if ALG_SHAKE256_256
            case TPM_ALG_SHAKE256_256:
#   endif  // ALG_SHAKE256_256
#   if ALG_SHAKE256_512
            case TPM_ALG_SHAKE256_512:
#   endif  // ALG_SHAKE256_512
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_HASH;
                break;
        }
    }
    return result;
}
#   if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_HASH_Marshal(TPMI_ALG_HASH* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#   endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_ASYM Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_ASYM_Unmarshal(TPMI_ALG_ASYM* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
```

```
        {
#   if ALG_RSA
            case TPM_ALG_RSA:
#   endif  // ALG_RSA
#   if ALG_ECC
            case TPM_ALG_ECC:
#   endif  // ALG_ECC
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_ASYMMETRIC;
                break;
        }
    }
    return result;
}
#   if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_ASYM_Marshal(TPMI_ALG_ASYM* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#   endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SYM Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SYM_Unmarshal(TPMI_ALG_SYM* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#   if ALG_AES
            case TPM_ALG_AES:
#   endif  // ALG_AES
#   if ALG_XOR
            case TPM_ALG_XOR:
#   endif  // ALG_XOR
#   if ALG_SM4
            case TPM_ALG_SM4:
#   endif  // ALG_SM4
#   if ALG_CAMELLIA
            case TPM_ALG_CAMELLIA:
#   endif  // ALG_CAMELLIA
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_SYMMETRIC;
                break;
        }
    }
    return result;
}
#   if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SYM_Marshal(TPMI_ALG_SYM* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#   endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SYM_OBJECT Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SYM_OBJECT_Unmarshal(
```

```
    TPMI_ALG_SYM_OBJECT* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#   if ALG_AES
            case TPM_ALG_AES:
#   endif  // ALG_AES
#   if ALG_SM4
            case TPM_ALG_SM4:
#   endif  // ALG_SM4
#   if ALG_CAMELLIA
            case TPM_ALG_CAMELLIA:
#   endif  // ALG_CAMELLIA
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_SYMMETRIC;
                break;
        }
    }
    return result;
}
#   if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SYM_OBJECT_Marshal(TPMI_ALG_SYM_OBJECT* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#   endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SYM_MODE Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SYM_MODE_Unmarshal(
    TPMI_ALG_SYM_MODE* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#   if ALG_CMAC
            case TPM_ALG_CMAC:
#   endif  // ALG_CMAC
#   if ALG_CTR
            case TPM_ALG_CTR:
#   endif  // ALG_CTR
#   if ALG_OFB
            case TPM_ALG_OFB:
#   endif  // ALG_OFB
#   if ALG_CBC
            case TPM_ALG_CBC:
#   endif  // ALG_CBC
#   if ALG_CFB
            case TPM_ALG_CFB:
#   endif  // ALG_CFB
#   if ALG_ECB
            case TPM_ALG_ECB:
#   endif  // ALG_ECB
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
```

```
                    result = TPM_RC_MODE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SYM_MODE_Marshal(TPMI_ALG_SYM_MODE* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_KDF Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_KDF_Unmarshal(TPMI_ALG_KDF* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_MGF1
            case TPM_ALG_MGF1:
#  endif  // ALG_MGF1
#  if ALG_KDF1_SP800_56A
            case TPM_ALG_KDF1_SP800_56A:
#  endif  // ALG_KDF1_SP800_56A
#  if ALG_KDF2
            case TPM_ALG_KDF2:
#  endif  // ALG_KDF2
#  if ALG_KDF1_SP800_108
            case TPM_ALG_KDF1_SP800_108:
#  endif  // ALG_KDF1_SP800_108
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_KDF;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_KDF_Marshal(TPMI_ALG_KDF* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_SIG_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_SIG_SCHEME_Unmarshal(
    TPMI_ALG_SIG_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_HMAC
            case TPM_ALG_HMAC:
```

```
#  endif  // ALG_HMAC
#  if ALG_RSASSA
            case TPM_ALG_RSASSA:
#  endif  // ALG_RSASSA
#  if ALG_RSAPSS
            case TPM_ALG_RSAPSS:
#  endif  // ALG_RSAPSS
#  if ALG_ECDSA
            case TPM_ALG_ECDSA:
#  endif  // ALG_ECDSA
#  if ALG_ECDAA
            case TPM_ALG_ECDAA:
#  endif  // ALG_ECDAA
#  if ALG_SM2
            case TPM_ALG_SM2:
#  endif  // ALG_SM2
#  if ALG_ECSCHNORR
            case TPM_ALG_ECSCHNORR:
#  endif  // ALG_ECSCHNORR
#  if ALG_EDDSA
            case TPM_ALG_EDDSA:
#  endif  // ALG_EDDSA
#  if ALG_EDDSA_PH
            case TPM_ALG_EDDSA_PH:
#  endif  // ALG_EDDSA_PH
#  if ALG_LMS
            case TPM_ALG_LMS:
#  endif  // ALG_LMS
#  if ALG_XMSS
            case TPM_ALG_XMSS:
#  endif  // ALG_XMSS
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_SCHEME;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_SIG_SCHEME_Marshal(TPMI_ALG_SIG_SCHEME* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ECC_KEY_EXCHANGE Type" (Part 2: Structures)
TPM_RC
TPMI_ECC_KEY_EXCHANGE_Unmarshal(
    TPMI_ECC_KEY_EXCHANGE* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_ECDH
            case TPM_ALG_ECDH:
#  endif  // ALG_ECDH
#  if ALG_SM2
            case TPM_ALG_SM2:
#  endif  // ALG_SM2
#  if ALG_ECMQV
```

```
            case TPM_ALG_ECMQV:
#  endif  // ALG_ECMQV
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_SCHEME;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ECC_KEY_EXCHANGE_Marshal(
    TPMI_ECC_KEY_EXCHANGE* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ST_COMMAND_TAG Type" (Part 2: Structures)
TPM_RC
TPMI_ST_COMMAND_TAG_Unmarshal(TPMI_ST_COMMAND_TAG* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_ST_Unmarshal((TPM_ST*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_ST_NO_SESSIONS:
            case TPM_ST_SESSIONS:
                break;
            default:
                result = TPM_RC_BAD_TAG;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ST_COMMAND_TAG_Marshal(TPMI_ST_COMMAND_TAG* source, BYTE** buffer, INT32* size)
{
    return TPM_ST_Marshal((TPM_ST*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_MAC_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_MAC_SCHEME_Unmarshal(
    TPMI_ALG_MAC_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_SHA1
            case TPM_ALG_SHA1:
#  endif  // ALG_SHA1
#  if ALG_SHA256
            case TPM_ALG_SHA256:
#  endif  // ALG_SHA256
#  if ALG_SHA384
```

```
            case TPM_ALG_SHA384:
#   endif  // ALG_SHA384
#   if ALG_SHA512
            case TPM_ALG_SHA512:
#   endif  // ALG_SHA512
#   if ALG_SHA256_192
            case TPM_ALG_SHA256_192:
#   endif  // ALG_SHA256_192
#   if ALG_SM3_256
            case TPM_ALG_SM3_256:
#   endif  // ALG_SM3_256
#   if ALG_SHA3_256
            case TPM_ALG_SHA3_256:
#   endif  // ALG_SHA3_256
#   if ALG_SHA3_384
            case TPM_ALG_SHA3_384:
#   endif  // ALG_SHA3_384
#   if ALG_SHA3_512
            case TPM_ALG_SHA3_512:
#   endif  // ALG_SHA3_512
#   if ALG_SHAKE256_192
            case TPM_ALG_SHAKE256_192:
#   endif  // ALG_SHAKE256_192
#   if ALG_SHAKE256_256
            case TPM_ALG_SHAKE256_256:
#   endif  // ALG_SHAKE256_256
#   if ALG_SHAKE256_512
            case TPM_ALG_SHAKE256_512:
#   endif  // ALG_SHAKE256_512
#   if ALG_CMAC
            case TPM_ALG_CMAC:
#   endif  // ALG_CMAC
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_SYMMETRIC;
                break;
        }
    }
    return result;
}
#   if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_MAC_SCHEME_Marshal(TPMI_ALG_MAC_SCHEME* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#   endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ALG_CIPHER_MODE Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_CIPHER_MODE_Unmarshal(
    TPMI_ALG_CIPHER_MODE* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#   if ALG_CTR
            case TPM_ALG_CTR:
#   endif  // ALG_CTR
#   if ALG_OFB
            case TPM_ALG_OFB:
#   endif  // ALG_OFB
```

```
#  if ALG_CBC
            case TPM_ALG_CBC:
#  endif  // ALG_CBC
#  if ALG_CFB
            case TPM_ALG_CFB:
#  endif  // ALG_CFB
#  if ALG_ECB
            case TPM_ALG_ECB:
#  endif  // ALG_ECB
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_MODE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_CIPHER_MODE_Marshal(TPMI_ALG_CIPHER_MODE* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_EMPTY Structure" (Part 2: Structures)
TPM_RC
TPMS_EMPTY_Unmarshal(TPMS_EMPTY* target, BYTE** buffer, INT32* size)
{
    // to prevent the compiler from complaining
    NOT_REFERENCED(target);
    NOT_REFERENCED(buffer);
    NOT_REFERENCED(size);
    return TPM_RC_SUCCESS;
}
UINT16
TPMS_EMPTY_Marshal(TPMS_EMPTY* source, BYTE** buffer, INT32* size)
{
    // to prevent the compiler from complaining
    NOT_REFERENCED(source);
    NOT_REFERENCED(buffer);
    NOT_REFERENCED(size);
    return 0;
}

// Table "Definition of TPMS_ALGORITHM_DESCRIPTION Structure" (Part 2: Structures)
UINT16
TPMS_ALGORITHM_DESCRIPTION_Marshal(
    TPMS_ALGORITHM_DESCRIPTION* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM_ALG_ID_Marshal((TPM_ALG_ID*)&(source->alg), buffer, size));
    result = (UINT16)(result
                      + TPMA_ALGORITHM_Marshal(
                          (TPMA_ALGORITHM*)&(source->attributes), buffer, size));
    return result;
}

// Table "Definition of TPMU_HA Union" (Part 2: Structures)
TPM_RC
TPMU_HA_Unmarshal(TPMU_HA* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
```

```
    {
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
#   if ALG_SHA1
        case TPM_ALG_SHA1:
            return BYTE_Array_Unmarshal(
                (BYTE*)&(target->sha1), buffer, size, (INT32)SHA1_DIGEST_SIZE);
#   endif  // ALG_SHA1
#   if ALG_SHA256
        case TPM_ALG_SHA256:
            return BYTE_Array_Unmarshal(
                (BYTE*)&(target->sha256), buffer, size, (INT32)SHA256_DIGEST_SIZE);
#   endif  // ALG_SHA256
#   if ALG_SHA256_192
        case TPM_ALG_SHA256_192:
            return BYTE_Array_Unmarshal((BYTE*)&(target->sha256_192),
                                        buffer,
                                        size,
                                        (INT32)SHA256_192_DIGEST_SIZE);
#   endif  // ALG_SHA256_192
#   if ALG_SHA3_256
        case TPM_ALG_SHA3_256:
            return BYTE_Array_Unmarshal((BYTE*)&(target->sha3_256),
                                        buffer,
                                        size,
                                        (INT32)SHA3_256_DIGEST_SIZE);
#   endif  // ALG_SHA3_256
#   if ALG_SHA3_384
        case TPM_ALG_SHA3_384:
            return BYTE_Array_Unmarshal((BYTE*)&(target->sha3_384),
                                        buffer,
                                        size,
                                        (INT32)SHA3_384_DIGEST_SIZE);
#   endif  // ALG_SHA3_384
#   if ALG_SHA3_512
        case TPM_ALG_SHA3_512:
            return BYTE_Array_Unmarshal((BYTE*)&(target->sha3_512),
                                        buffer,
                                        size,
                                        (INT32)SHA3_512_DIGEST_SIZE);
#   endif  // ALG_SHA3_512
#   if ALG_SHA384
        case TPM_ALG_SHA384:
            return BYTE_Array_Unmarshal(
                (BYTE*)&(target->sha384), buffer, size, (INT32)SHA384_DIGEST_SIZE);
#   endif  // ALG_SHA384
#   if ALG_SHA512
        case TPM_ALG_SHA512:
            return BYTE_Array_Unmarshal(
                (BYTE*)&(target->sha512), buffer, size, (INT32)SHA512_DIGEST_SIZE);
#   endif  // ALG_SHA512
#   if ALG_SHAKE256_192
        case TPM_ALG_SHAKE256_192:
            return BYTE_Array_Unmarshal((BYTE*)&(target->shake256_192),
                                        buffer,
                                        size,
                                        (INT32)SHAKE256_192_DIGEST_SIZE);
#   endif  // ALG_SHAKE256_192
#   if ALG_SHAKE256_256
        case TPM_ALG_SHAKE256_256:
            return BYTE_Array_Unmarshal((BYTE*)&(target->shake256_256),
                                        buffer,
                                        size,
                                        (INT32)SHAKE256_256_DIGEST_SIZE);
#   endif  // ALG_SHAKE256_256
#   if ALG_SHAKE256_512
```

```
        case TPM_ALG_SHAKE256_512:
            return BYTE_Array_Unmarshal((BYTE*)&(target->shake256_512),
                                        buffer,
                                        size,
                                        (INT32)SHAKE256_512_DIGEST_SIZE);
#   endif  // ALG_SHAKE256_512
#   if ALG_SM3_256
        case TPM_ALG_SM3_256:
            return BYTE_Array_Unmarshal(
                (BYTE*)&(target->sm3_256), buffer, size, (INT32)SM3_256_DIGEST_SIZE);
#   endif  // ALG_SM3_256
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_HA_Marshal(TPMU_HA* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_SHA1
        case TPM_ALG_SHA1:
            return BYTE_Array_Marshal(
                (BYTE*)&(source->sha1), buffer, size, (INT32)SHA1_DIGEST_SIZE);
#   endif  // ALG_SHA1
#   if ALG_SHA256
        case TPM_ALG_SHA256:
            return BYTE_Array_Marshal(
                (BYTE*)&(source->sha256), buffer, size, (INT32)SHA256_DIGEST_SIZE);
#   endif  // ALG_SHA256
#   if ALG_SHA256_192
        case TPM_ALG_SHA256_192:
            return BYTE_Array_Marshal((BYTE*)&(source->sha256_192),
                                      buffer,
                                      size,
                                      (INT32)SHA256_192_DIGEST_SIZE);
#   endif  // ALG_SHA256_192
#   if ALG_SHA3_256
        case TPM_ALG_SHA3_256:
            return BYTE_Array_Marshal((BYTE*)&(source->sha3_256),
                                      buffer,
                                      size,
                                      (INT32)SHA3_256_DIGEST_SIZE);
#   endif  // ALG_SHA3_256
#   if ALG_SHA3_384
        case TPM_ALG_SHA3_384:
            return BYTE_Array_Marshal((BYTE*)&(source->sha3_384),
                                      buffer,
                                      size,
                                      (INT32)SHA3_384_DIGEST_SIZE);
#   endif  // ALG_SHA3_384
#   if ALG_SHA3_512
        case TPM_ALG_SHA3_512:
            return BYTE_Array_Marshal((BYTE*)&(source->sha3_512),
                                      buffer,
                                      size,
                                      (INT32)SHA3_512_DIGEST_SIZE);
#   endif  // ALG_SHA3_512
#   if ALG_SHA384
        case TPM_ALG_SHA384:
            return BYTE_Array_Marshal(
                (BYTE*)&(source->sha384), buffer, size, (INT32)SHA384_DIGEST_SIZE);
#   endif  // ALG_SHA384
#   if ALG_SHA512
        case TPM_ALG_SHA512:
            return BYTE_Array_Marshal(
                (BYTE*)&(source->sha512), buffer, size, (INT32)SHA512_DIGEST_SIZE);
```

```
#  endif  // ALG_SHA512
#  if ALG_SHAKE256_192
        case TPM_ALG_SHAKE256_192:
            return BYTE_Array_Marshal((BYTE*)&(source->shake256_192),
                                      buffer,
                                      size,
                                      (INT32)SHAKE256_192_DIGEST_SIZE);
#  endif  // ALG_SHAKE256_192
#  if ALG_SHAKE256_256
        case TPM_ALG_SHAKE256_256:
            return BYTE_Array_Marshal((BYTE*)&(source->shake256_256),
                                      buffer,
                                      size,
                                      (INT32)SHAKE256_256_DIGEST_SIZE);
#  endif  // ALG_SHAKE256_256
#  if ALG_SHAKE256_512
        case TPM_ALG_SHAKE256_512:
            return BYTE_Array_Marshal((BYTE*)&(source->shake256_512),
                                      buffer,
                                      size,
                                      (INT32)SHAKE256_512_DIGEST_SIZE);
#  endif  // ALG_SHAKE256_512
#  if ALG_SM3_256
        case TPM_ALG_SM3_256:
            return BYTE_Array_Marshal(
                (BYTE*)&(source->sm3_256), buffer, size, (INT32)SM3_256_DIGEST_SIZE);
#  endif  // ALG_SM3_256
    }
    return 0;
}

// Table "Definition of TPMT_HA Structure" (Part 2: Structures)
TPM_RC
TPMT_HA_Unmarshal(TPMT_HA* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_HASH_Unmarshal(
        (TPMI_ALG_HASH*)&(target->hashAlg), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_HA_Unmarshal(
            (TPMU_HA*)&(target->digest), buffer, size, (UINT32)target->hashAlg);
    return result;
}
UINT16
TPMT_HA_Marshal(TPMT_HA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
                          (TPMI_ALG_HASH*)&(source->hashAlg), buffer, size));
    result        = (UINT16)(result
                      + TPMU_HA_Marshal((TPMU_HA*)&(source->digest),
                                        buffer,
                                        size,
                                        (UINT32)source->hashAlg));
    return result;
}

// Table "Definition of TPM2B_DIGEST Structure" (Part 2: Structures)
TPM_RC
TPM2B_DIGEST_Unmarshal(TPM2B_DIGEST* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMU_HA)))
        result = TPM_RC_SIZE;
```

```
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_DIGEST_Marshal(TPM2B_DIGEST* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_DATA_Unmarshal(TPM2B_DATA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMT_HA)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_DATA_Marshal(TPM2B_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of Types for TPM2B_NONCE" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM2B_NONCE_Unmarshal(TPM2B_NONCE* target, BYTE** buffer, INT32* size)
{
    return TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)target, buffer, size);
}
UINT16
TPM2B_NONCE_Marshal(TPM2B_NONCE* source, BYTE** buffer, INT32* size)
{
    return TPM2B_DIGEST_Marshal((TPM2B_DIGEST*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES
```

```
// Table "Definition of Types for TPM2B_AUTH" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM2B_AUTH_Unmarshal(TPM2B_AUTH* target, BYTE** buffer, INT32* size)
{
    return TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)target, buffer, size);
}
UINT16
TPM2B_AUTH_Marshal(TPM2B_AUTH* source, BYTE** buffer, INT32* size)
{
    return TPM2B_DIGEST_Marshal((TPM2B_DIGEST*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for TPM2B_OPERAND" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPM2B_OPERAND_Unmarshal(TPM2B_OPERAND* target, BYTE** buffer, INT32* size)
{
    return TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)target, buffer, size);
}
UINT16
TPM2B_OPERAND_Marshal(TPM2B_OPERAND* source, BYTE** buffer, INT32* size)
{
    return TPM2B_DIGEST_Marshal((TPM2B_DIGEST*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM2B_EVENT Structure" (Part 2: Structures)
TPM_RC
TPM2B_EVENT_Unmarshal(TPM2B_EVENT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > 1024))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_EVENT_Marshal(TPM2B_EVENT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_MAX_BUFFER Structure" (Part 2: Structures)
TPM_RC
TPM2B_MAX_BUFFER_Unmarshal(TPM2B_MAX_BUFFER* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > MAX_DIGEST_BUFFER))
        result = TPM_RC_SIZE;
```

```
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_MAX_BUFFER_Marshal(TPM2B_MAX_BUFFER* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_MAX_NV_BUFFER Structure" (Part 2: Structures)
TPM_RC
TPM2B_MAX_NV_BUFFER_Unmarshal(TPM2B_MAX_NV_BUFFER* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > MAX_NV_BUFFER_SIZE))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_MAX_NV_BUFFER_Marshal(TPM2B_MAX_NV_BUFFER* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_TIMEOUT Structure" (Part 2: Structures)
TPM_RC
TPM2B_TIMEOUT_Unmarshal(TPM2B_TIMEOUT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(UINT64)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
```

```
TPM2B_TIMEOUT_Marshal(TPM2B_TIMEOUT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_IV Structure" (Part 2: Structures)
TPM_RC
TPM2B_IV_Unmarshal(TPM2B_IV* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > MAX_SYM_BLOCK_SIZE))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_IV_Marshal(TPM2B_IV* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_VENDOR_PROPERTY Structure" (Part 2: Structures)
TPM_RC
TPM2B_VENDOR_PROPERTY_Unmarshal(
    TPM2B_VENDOR_PROPERTY* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > 512))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_VENDOR_PROPERTY_Marshal(
    TPM2B_VENDOR_PROPERTY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
```

```
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_NAME Structure" (Part 2: Structures)
TPM_RC
TPM2B_NAME_Unmarshal(TPM2B_NAME* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMU_NAME)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.name), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_NAME_Marshal(TPM2B_NAME* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result =
        (UINT16)(result
                 + BYTE_Array_Marshal(
                     (BYTE*)&(source->t.name), buffer, size, (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_PCR_SELECT Structure" (Part 2: Structures)
TPM_RC
TPMS_PCR_SELECT_Unmarshal(TPMS_PCR_SELECT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT8_Unmarshal((UINT8*)&(target->sizeofSelect), buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((target->sizeofSelect < PCR_SELECT_MIN)
           || (target->sizeofSelect > PCR_SELECT_MAX)))
        result = TPM_RC_VALUE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->pcrSelect), buffer, size, (INT32)target->sizeofSelect);
    return result;
}
UINT16
TPMS_PCR_SELECT_Marshal(TPMS_PCR_SELECT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + UINT8_Marshal((UINT8*)&(source->sizeofSelect), buffer, size));
    result        = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->pcrSelect),
                                           buffer,
                                           size,
```

```
                                            (INT32)source->sizeofSelect));
    return result;
}

// Table "Definition of TPMS_PCR_SELECTION Structure" (Part 2: Structures)
TPM_RC
TPMS_PCR_SELECTION_Unmarshal(TPMS_PCR_SELECTION* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result =
        TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH*)&(target->hash), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
        result = UINT8_Unmarshal((UINT8*)&(target->sizeofSelect), buffer, size);
    if((result == TPM_RC_SUCCESS)
       && ((target->sizeofSelect < PCR_SELECT_MIN)
           || (target->sizeofSelect > PCR_SELECT_MAX)))
        result = TPM_RC_VALUE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->pcrSelect), buffer, size, (INT32)target->sizeofSelect);
    return result;
}
UINT16
TPMS_PCR_SELECTION_Marshal(TPMS_PCR_SELECTION* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
                          (TPMI_ALG_HASH*)&(source->hash), buffer, size));
    result        = (UINT16)(result
                      + UINT8_Marshal((UINT8*)&(source->sizeofSelect), buffer, size));
    result        = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->pcrSelect),
                                           buffer,
                                           size,
                                           (INT32)source->sizeofSelect));
    return result;
}

// Table "Definition of TPMT_TK_CREATION Structure" (Part 2: Structures)
TPM_RC
TPMT_TK_CREATION_Unmarshal(TPMT_TK_CREATION* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_ST_Unmarshal((TPM_ST*)&(target->tag), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->tag != TPM_ST_CREATION))
        result = TPM_RC_TAG;
    if(result == TPM_RC_SUCCESS)
        result = TPMI_RH_HIERARCHY_Unmarshal(
            (TPMI_RH_HIERARCHY*)&(target->hierarchy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)&(target->digest), buffer, size);
    return result;
}
UINT16
TPMT_TK_CREATION_Marshal(TPMT_TK_CREATION* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result = (UINT16)(result + TPM_ST_Marshal((TPM_ST*)&(source->tag), buffer, size));
    result = (UINT16)(result
                      + TPMI_RH_HIERARCHY_Marshal(
                          (TPMI_RH_HIERARCHY*)&(source->hierarchy), buffer, size));
    result = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->digest), buffer, size));
```

```
    return result;
}

// Table "Definition of TPMT_TK_VERIFIED Structure" (Part 2: Structures)
TPM_RC
TPMT_TK_VERIFIED_Unmarshal(TPMT_TK_VERIFIED* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_ST_Unmarshal((TPM_ST*)&(target->tag), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->tag != TPM_ST_VERIFIED))
        result = TPM_RC_TAG;
    if(result == TPM_RC_SUCCESS)
        result = TPMI_RH_HIERARCHY_Unmarshal(
            (TPMI_RH_HIERARCHY*)&(target->hierarchy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)&(target->digest), buffer, size);
    return result;
}
UINT16
TPMT_TK_VERIFIED_Marshal(TPMT_TK_VERIFIED* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result = (UINT16)(result + TPM_ST_Marshal((TPM_ST*)&(source->tag), buffer, size));
    result = (UINT16)(result
                      + TPMI_RH_HIERARCHY_Marshal(
                          (TPMI_RH_HIERARCHY*)&(source->hierarchy), buffer, size));
    result = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->digest), buffer, size));
    return result;
}

// Table "Definition of TPMT_TK_AUTH Structure" (Part 2: Structures)
TPM_RC
TPMT_TK_AUTH_Unmarshal(TPMT_TK_AUTH* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_ST_Unmarshal((TPM_ST*)&(target->tag), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->tag != TPM_ST_AUTH_SIGNED)
       && (target->tag != TPM_ST_AUTH_SECRET))
        result = TPM_RC_TAG;
    if(result == TPM_RC_SUCCESS)
        result = TPMI_RH_HIERARCHY_Unmarshal(
            (TPMI_RH_HIERARCHY*)&(target->hierarchy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)&(target->digest), buffer, size);
    return result;
}
UINT16
TPMT_TK_AUTH_Marshal(TPMT_TK_AUTH* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result = (UINT16)(result + TPM_ST_Marshal((TPM_ST*)&(source->tag), buffer, size));
    result = (UINT16)(result
                      + TPMI_RH_HIERARCHY_Marshal(
                          (TPMI_RH_HIERARCHY*)&(source->hierarchy), buffer, size));
    result = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->digest), buffer, size));
    return result;
}

// Table "Definition of TPMT_TK_HASHCHECK Structure" (Part 2: Structures)
TPM_RC
```

```
TPMT_TK_HASHCHECK_Unmarshal(TPMT_TK_HASHCHECK* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_ST_Unmarshal((TPM_ST*)&(target->tag), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->tag != TPM_ST_HASHCHECK))
        result = TPM_RC_TAG;
    if(result == TPM_RC_SUCCESS)
        result = TPMI_RH_HIERARCHY_Unmarshal(
            (TPMI_RH_HIERARCHY*)&(target->hierarchy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)&(target->digest), buffer, size);
    return result;
}
UINT16
TPMT_TK_HASHCHECK_Marshal(TPMT_TK_HASHCHECK* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result = (UINT16)(result + TPM_ST_Marshal((TPM_ST*)&(source->tag), buffer, size));
    result = (UINT16)(result
                      + TPMI_RH_HIERARCHY_Marshal(
                          (TPMI_RH_HIERARCHY*)&(source->hierarchy), buffer, size));
    result = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->digest), buffer, size));
    return result;
}

// Table "Definition of TPMS_ALG_PROPERTY Structure" (Part 2: Structures)
UINT16
TPMS_ALG_PROPERTY_Marshal(TPMS_ALG_PROPERTY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM_ALG_ID_Marshal((TPM_ALG_ID*)&(source->alg), buffer, size));
    result = (UINT16)(result
                      + TPMA_ALGORITHM_Marshal(
                          (TPMA_ALGORITHM*)&(source->algProperties), buffer, size));
    return result;
}

// Table "Definition of TPMS_TAGGED_PROPERTY Structure" (Part 2: Structures)
UINT16
TPMS_TAGGED_PROPERTY_Marshal(TPMS_TAGGED_PROPERTY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + TPM_PT_Marshal((TPM_PT*)&(source->property), buffer, size));
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->value), buffer, size));
    return result;
}

// Table "Definition of TPMS_TAGGED_PCR_SELECT Structure" (Part 2: Structures)
UINT16
TPMS_TAGGED_PCR_SELECT_Marshal(
    TPMS_TAGGED_PCR_SELECT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM_PT_PCR_Marshal((TPM_PT_PCR*)&(source->tag), buffer, size));
    result = (UINT16)(result
                      + UINT8_Marshal((UINT8*)&(source->sizeofSelect), buffer, size));
    result = (UINT16)(result
```

```
                    + BYTE_Array_Marshal((BYTE*)&(source->pcrSelect),
                                         buffer,
                                         size,
                                         (INT32)source->sizeofSelect));
    return result;
}

// Table "Definition of TPMS_TAGGED_POLICY Structure" (Part 2: Structures)
UINT16
TPMS_TAGGED_POLICY_Marshal(TPMS_TAGGED_POLICY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM_HANDLE_Marshal((TPM_HANDLE*)&(source->handle), buffer, size));
    result =
        (UINT16)(result
                 + TPMT_HA_Marshal((TPMT_HA*)&(source->policyHash), buffer, size));
    return result;
}

// Table "Definition of TPMS_ACT_DATA Structure" (Part 2: Structures)
UINT16
TPMS_ACT_DATA_Marshal(TPMS_ACT_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM_HANDLE_Marshal((TPM_HANDLE*)&(source->handle), buffer, size));
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->timeout), buffer, size));
    result =
        (UINT16)(result
                 + TPMA_ACT_Marshal((TPMA_ACT*)&(source->attributes), buffer, size));
    return result;
}

// Table "Definition of TPML_CC Structure" (Part 2: Structures)
TPM_RC
TPML_CC_Unmarshal(TPML_CC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)&(target->count), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->count > MAX_CAP_CC))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = TPM_CC_Array_Unmarshal(
            (TPM_CC*)&(target->commandCodes), buffer, size, (INT32)target->count);
    return result;
}
UINT16
TPML_CC_Marshal(TPML_CC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                    + TPM_CC_Array_Marshal((TPM_CC*)&(source->commandCodes),
                                           buffer,
                                           size,
                                           (INT32)source->count));
    return result;
}

// Table "Definition of TPML_CCA Structure" (Part 2: Structures)
UINT16
```

```
TPML_CCA_Marshal(TPML_CCA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                      + TPMA_CC_Array_Marshal((TPMA_CC*)&(source->commandAttributes),
                                              buffer,
                                              size,
                                              (INT32)source->count));
    return result;
}

// Table "Definition of TPML_ALG Structure" (Part 2: Structures)
TPM_RC
TPML_ALG_Unmarshal(TPML_ALG* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)&(target->count), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->count > MAX_ALG_LIST_SIZE))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = TPM_ALG_ID_Array_Unmarshal(
            (TPM_ALG_ID*)&(target->algorithms), buffer, size, (INT32)target->count);
    return result;
}
UINT16
TPML_ALG_Marshal(TPML_ALG* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                      + TPM_ALG_ID_Array_Marshal((TPM_ALG_ID*)&(source->algorithms),
                                                 buffer,
                                                 size,
                                                 (INT32)source->count));
    return result;
}

// Table "Definition of TPML_HANDLE Structure" (Part 2: Structures)
UINT16
TPML_HANDLE_Marshal(TPML_HANDLE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                      + TPM_HANDLE_Array_Marshal((TPM_HANDLE*)&(source->handle),
                                                 buffer,
                                                 size,
                                                 (INT32)source->count));
    return result;
}

// Table "Definition of TPML_DIGEST Structure" (Part 2: Structures)
TPM_RC
TPML_DIGEST_Unmarshal(TPML_DIGEST* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)&(target->count), buffer, size);
    if((result == TPM_RC_SUCCESS) && ((target->count < 2) || (target->count > 8)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_DIGEST_Array_Unmarshal(
            (TPM2B_DIGEST*)&(target->digests), buffer, size, (INT32)target->count);
```

```
    return result;
}
UINT16
TPML_DIGEST_Marshal(TPML_DIGEST* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                      + TPM2B_DIGEST_Array_Marshal((TPM2B_DIGEST*)&(source->digests),
                                                   buffer,
                                                   size,
                                                   (INT32)source->count));
    return result;
}

// Table "Definition of TPML_DIGEST_VALUES Structure" (Part 2: Structures)
TPM_RC
TPML_DIGEST_VALUES_Unmarshal(TPML_DIGEST_VALUES* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)&(target->count), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->count > HASH_COUNT))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = TPMT_HA_Array_Unmarshal(
            (TPMT_HA*)&(target->digests), buffer, size, 0, (INT32)target->count);
    return result;
}
UINT16
TPML_DIGEST_VALUES_Marshal(TPML_DIGEST_VALUES* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                      + TPMT_HA_Array_Marshal((TPMT_HA*)&(source->digests),
                                              buffer,
                                              size,
                                              (INT32)source->count));
    return result;
}

// Table "Definition of TPML_PCR_SELECTION Structure" (Part 2: Structures)
TPM_RC
TPML_PCR_SELECTION_Unmarshal(TPML_PCR_SELECTION* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)&(target->count), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->count > HASH_COUNT))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = TPMS_PCR_SELECTION_Array_Unmarshal(
            (TPMS_PCR_SELECTION*)&(target->pcrSelections),
            buffer,
            size,
            (INT32)target->count);
    return result;
}
UINT16
TPML_PCR_SELECTION_Marshal(TPML_PCR_SELECTION* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
```

```
                        + TPMS_PCR_SELECTION_Array_Marshal(
                            (TPMS_PCR_SELECTION*)&(source->pcrSelections),
                            buffer,
                            size,
                            (INT32)source->count));
    return result;
}

// Table "Definition of TPML_ALG_PROPERTY Structure" (Part 2: Structures)
UINT16
TPML_ALG_PROPERTY_Marshal(TPML_ALG_PROPERTY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                        + TPMS_ALG_PROPERTY_Array_Marshal(
                            (TPMS_ALG_PROPERTY*)&(source->algProperties),
                            buffer,
                            size,
                            (INT32)source->count));
    return result;
}

// Table "Definition of TPML_TAGGED_TPM_PROPERTY Structure" (Part 2: Structures)
UINT16
TPML_TAGGED_TPM_PROPERTY_Marshal(
    TPML_TAGGED_TPM_PROPERTY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                        + TPMS_TAGGED_PROPERTY_Array_Marshal(
                            (TPMS_TAGGED_PROPERTY*)&(source->tpmProperty),
                            buffer,
                            size,
                            (INT32)source->count));
    return result;
}

// Table "Definition of TPML_TAGGED_PCR_PROPERTY Structure" (Part 2: Structures)
UINT16
TPML_TAGGED_PCR_PROPERTY_Marshal(
    TPML_TAGGED_PCR_PROPERTY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                        + TPMS_TAGGED_PCR_SELECT_Array_Marshal(
                            (TPMS_TAGGED_PCR_SELECT*)&(source->pcrProperty),
                            buffer,
                            size,
                            (INT32)source->count));
    return result;
}

// Table "Definition of TPML_ECC_CURVE Structure" (Part 2: Structures)
#  if ALG_ECC
UINT16
TPML_ECC_CURVE_Marshal(TPML_ECC_CURVE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
```

```
    result =
        (UINT16)(result
                + TPM_ECC_CURVE_Array_Marshal((TPM_ECC_CURVE*)&(source->eccCurves),
                                              buffer,
                                              size,
                                              (INT32)source->count));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of TPML_TAGGED_POLICY Structure" (Part 2: Structures)
UINT16
TPML_TAGGED_POLICY_Marshal(TPML_TAGGED_POLICY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                      + TPMS_TAGGED_POLICY_Array_Marshal(
                          (TPMS_TAGGED_POLICY*)&(source->policies),
                          buffer,
                          size,
                          (INT32)source->count));
    return result;
}

// Table "Definition of TPML_ACT_DATA Structure" (Part 2: Structures)
UINT16
TPML_ACT_DATA_Marshal(TPML_ACT_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result =
        (UINT16)(result
                + TPMS_ACT_DATA_Array_Marshal((TPMS_ACT_DATA*)&(source->actData),
                                              buffer,
                                              size,
                                              (INT32)source->count));
    return result;
}

// Table "Definition of TPML_VENDOR_PROPERTY Structure" (Part 2: Structures)
TPM_RC
TPML_VENDOR_PROPERTY_Unmarshal(
    TPML_VENDOR_PROPERTY* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)&(target->count), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->count > MAX_VENDOR_PROPERTY))
        result = TPM_RC_VALUE;
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_VENDOR_PROPERTY_Array_Unmarshal(
            (TPM2B_VENDOR_PROPERTY*)&(target->vendorData),
            buffer,
            size,
            (INT32)target->count);
    return result;
}
UINT16
TPML_VENDOR_PROPERTY_Marshal(TPML_VENDOR_PROPERTY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
```

```
                        + TPM2B_VENDOR_PROPERTY_Array_Marshal(
                            (TPM2B_VENDOR_PROPERTY*)&(source->vendorData),
                            buffer,
                            size,
                            (INT32)source->count));
    return result;
}

// Table "Definition of TPMU_CAPABILITIES Union" (Part 2: Structures)
UINT16
TPMU_CAPABILITIES_Marshal(
    TPMU_CAPABILITIES* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
        case TPM_CAP_ALGS:
            return TPML_ALG_PROPERTY_Marshal(
                (TPML_ALG_PROPERTY*)&(source->algorithms), buffer, size);
        case TPM_CAP_HANDLES:
            return TPML_HANDLE_Marshal(
                (TPML_HANDLE*)&(source->handles), buffer, size);
        case TPM_CAP_COMMANDS:
            return TPML_CCA_Marshal((TPML_CCA*)&(source->command), buffer, size);
        case TPM_CAP_PP_COMMANDS:
            return TPML_CC_Marshal((TPML_CC*)&(source->ppCommands), buffer, size);
        case TPM_CAP_AUDIT_COMMANDS:
            return TPML_CC_Marshal((TPML_CC*)&(source->auditCommands), buffer, size);
        case TPM_CAP_PCRS:
            return TPML_PCR_SELECTION_Marshal(
                (TPML_PCR_SELECTION*)&(source->assignedPCR), buffer, size);
        case TPM_CAP_TPM_PROPERTIES:
            return TPML_TAGGED_TPM_PROPERTY_Marshal(
                (TPML_TAGGED_TPM_PROPERTY*)&(source->tpmProperties), buffer, size);
        case TPM_CAP_PCR_PROPERTIES:
            return TPML_TAGGED_PCR_PROPERTY_Marshal(
                (TPML_TAGGED_PCR_PROPERTY*)&(source->pcrProperties), buffer, size);
#   if ALG_ECC
        case TPM_CAP_ECC_CURVES:
            return TPML_ECC_CURVE_Marshal(
                (TPML_ECC_CURVE*)&(source->eccCurves), buffer, size);
#   endif  // ALG_ECC
        case TPM_CAP_AUTH_POLICIES:
            return TPML_TAGGED_POLICY_Marshal(
                (TPML_TAGGED_POLICY*)&(source->authPolicies), buffer, size);
        case TPM_CAP_ACT:
            return TPML_ACT_DATA_Marshal(
                (TPML_ACT_DATA*)&(source->actData), buffer, size);
    }
    return 0;
}

// Table "Definition of TPMS_CAPABILITY_DATA Structure" (Part 2: Structures)
UINT16
TPMS_CAPABILITY_DATA_Marshal(TPMS_CAPABILITY_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM_CAP_Marshal((TPM_CAP*)&(source->capability), buffer, size));
    result = (UINT16)(result
                      + TPMU_CAPABILITIES_Marshal((TPMU_CAPABILITIES*)&(source->data),
                                                  buffer,
                                                  size,
                                                  (UINT32)source->capability));
    return result;
}
```

```
// Defined in an additional Capabilities registry
TPM_RC
TPMU_SET_CAPABILITIES_Unmarshal(
    TPMU_SET_CAPABILITIES* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    NOT_REFERENCED(target);
    NOT_REFERENCED(buffer);
    NOT_REFERENCED(size);
    NOT_REFERENCED(selector);

    // No settable capabilities are currently defined in the reference code.
    return TPM_RC_SELECTOR;
}

// Table "Definition of TPMS_SET_CAPABILITY_DATA Structure" (Part 2: Structures)
TPM_RC
TPMS_SET_CAPABILITY_DATA_Unmarshal(
    TPMS_SET_CAPABILITY_DATA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_CAP_Unmarshal(&target->setCapability, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        result = TPMU_SET_CAPABILITIES_Unmarshal(
            &target->data, buffer, size, (UINT32)target->setCapability);
    }
    return result;
}

// Table "Definition of TPM2B_SET_CAPABILITY_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_SET_CAPABILITY_DATA_Unmarshal(
    TPM2B_SET_CAPABILITY_DATA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->size), buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // if size is zero, then the required structure is missing
        if(target->size == 0)
            result = TPM_RC_SIZE;
        else
        {
            INT32 startSize = *size;
            result          = TPMS_SET_CAPABILITY_DATA_Unmarshal(
                &target->setCapabilityData, buffer, size);
            if((result == TPM_RC_SUCCESS) && (target->size != (startSize - *size)))
                result = TPM_RC_SIZE;
        }
    }
    return result;
}

// Table "Definition of TPMS_CLOCK_INFO Structure" (Part 2: Structures)
TPM_RC
TPMS_CLOCK_INFO_Unmarshal(TPMS_CLOCK_INFO* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT64_Unmarshal((UINT64*)&(target->clock), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = UINT32_Unmarshal((UINT32*)&(target->resetCount), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = UINT32_Unmarshal((UINT32*)&(target->restartCount), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_YES_NO_Unmarshal((TPMI_YES_NO*)&(target->safe), buffer, size);
```

```
    return result;
}
UINT16
TPMS_CLOCK_INFO_Marshal(TPMS_CLOCK_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT64_Marshal((UINT64*)&(source->clock), buffer, size));
    result = (UINT16)(result
                      + UINT32_Marshal((UINT32*)&(source->resetCount), buffer, size));
    result =
        (UINT16)(result
                 + UINT32_Marshal((UINT32*)&(source->restartCount), buffer, size));
    result =
        (UINT16)(result
                 + TPMI_YES_NO_Marshal((TPMI_YES_NO*)&(source->safe), buffer, size));
    return result;
}

// Table "Definition of TPMS_TIME_INFO Structure" (Part 2: Structures)
TPM_RC
TPMS_TIME_INFO_Unmarshal(TPMS_TIME_INFO* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT64_Unmarshal((UINT64*)&(target->time), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMS_CLOCK_INFO_Unmarshal(
            (TPMS_CLOCK_INFO*)&(target->clockInfo), buffer, size);
    return result;
}
UINT16
TPMS_TIME_INFO_Marshal(TPMS_TIME_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT64_Marshal((UINT64*)&(source->time), buffer, size));
    result = (UINT16)(result
                      + TPMS_CLOCK_INFO_Marshal(
                          (TPMS_CLOCK_INFO*)&(source->clockInfo), buffer, size));
    return result;
}

// Table "Definition of TPMS_TIME_ATTEST_INFO Structure" (Part 2: Structures)
UINT16
TPMS_TIME_ATTEST_INFO_Marshal(
    TPMS_TIME_ATTEST_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMS_TIME_INFO_Marshal(
                          (TPMS_TIME_INFO*)&(source->time), buffer, size));
    result =
        (UINT16)(result
                 + UINT64_Marshal((UINT64*)&(source->firmwareVersion), buffer, size));
    return result;
}

// Table "Definition of TPMS_CERTIFY_INFO Structure" (Part 2: Structures)
UINT16
TPMS_CERTIFY_INFO_Marshal(TPMS_CERTIFY_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM2B_NAME_Marshal((TPM2B_NAME*)&(source->name), buffer, size));
    result = (UINT16)(result
```

```
                        + TPM2B_NAME_Marshal(
                            (TPM2B_NAME*)&(source->qualifiedName), buffer, size));
    return result;
}

// Table "Definition of TPMS_QUOTE_INFO Structure" (Part 2: Structures)
UINT16
TPMS_QUOTE_INFO_Marshal(TPMS_QUOTE_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                        + TPML_PCR_SELECTION_Marshal(
                            (TPML_PCR_SELECTION*)&(source->pcrSelect), buffer, size));
    result        = (UINT16)(result
                        + TPM2B_DIGEST_Marshal(
                            (TPM2B_DIGEST*)&(source->pcrDigest), buffer, size));
    return result;
}

// Table "Definition of TPMS_COMMAND_AUDIT_INFO Structure" (Part 2: Structures)
UINT16
TPMS_COMMAND_AUDIT_INFO_Marshal(
    TPMS_COMMAND_AUDIT_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + UINT64_Marshal((UINT64*)&(source->auditCounter), buffer, size));
    result = (UINT16)(result
                        + TPM_ALG_ID_Marshal(
                            (TPM_ALG_ID*)&(source->digestAlg), buffer, size));
    result = (UINT16)(result
                        + TPM2B_DIGEST_Marshal(
                            (TPM2B_DIGEST*)&(source->auditDigest), buffer, size));
    result = (UINT16)(result
                        + TPM2B_DIGEST_Marshal(
                            (TPM2B_DIGEST*)&(source->commandDigest), buffer, size));
    return result;
}

// Table "Definition of TPMS_SESSION_AUDIT_INFO Structure" (Part 2: Structures)
UINT16
TPMS_SESSION_AUDIT_INFO_Marshal(
    TPMS_SESSION_AUDIT_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                        + TPMI_YES_NO_Marshal(
                            (TPMI_YES_NO*)&(source->exclusiveSession), buffer, size));
    result        = (UINT16)(result
                        + TPM2B_DIGEST_Marshal(
                            (TPM2B_DIGEST*)&(source->sessionDigest), buffer, size));
    return result;
}

// Table "Definition of TPMS_CREATION_INFO Structure" (Part 2: Structures)
UINT16
TPMS_CREATION_INFO_Marshal(TPMS_CREATION_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                        + TPM2B_NAME_Marshal(
                            (TPM2B_NAME*)&(source->objectName), buffer, size));
    result        = (UINT16)(result
                        + TPM2B_DIGEST_Marshal(
                            (TPM2B_DIGEST*)&(source->creationHash), buffer, size));
```

```
    return result;
}

// Table "Definition of TPMS_NV_CERTIFY_INFO Structure" (Part 2: Structures)
UINT16
TPMS_NV_CERTIFY_INFO_Marshal(TPMS_NV_CERTIFY_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPM2B_NAME_Marshal(
                          (TPM2B_NAME*)&(source->indexName), buffer, size));
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->offset), buffer, size));
    result = (UINT16)(result
                      + TPM2B_MAX_NV_BUFFER_Marshal(
                          (TPM2B_MAX_NV_BUFFER*)&(source->nvContents), buffer, size));
    return result;
}

// Table "Definition of TPMS_NV_DIGEST_CERTIFY_INFO Structure" (Part 2: Structures)
UINT16
TPMS_NV_DIGEST_CERTIFY_INFO_Marshal(
    TPMS_NV_DIGEST_CERTIFY_INFO* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPM2B_NAME_Marshal(
                          (TPM2B_NAME*)&(source->indexName), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->nvDigest), buffer, size));
    return result;
}

// Table "Definition of TPMI_ST_ATTEST Type" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ST_ATTEST_Marshal(TPMI_ST_ATTEST* source, BYTE** buffer, INT32* size)
{
    return TPM_ST_Marshal((TPM_ST*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_ATTEST Union" (Part 2: Structures)
UINT16
TPMU_ATTEST_Marshal(TPMU_ATTEST* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
        case TPM_ST_ATTEST_CERTIFY:
            return TPMS_CERTIFY_INFO_Marshal(
                (TPMS_CERTIFY_INFO*)&(source->certify), buffer, size);
        case TPM_ST_ATTEST_CREATION:
            return TPMS_CREATION_INFO_Marshal(
                (TPMS_CREATION_INFO*)&(source->creation), buffer, size);
        case TPM_ST_ATTEST_QUOTE:
            return TPMS_QUOTE_INFO_Marshal(
                (TPMS_QUOTE_INFO*)&(source->quote), buffer, size);
        case TPM_ST_ATTEST_COMMAND_AUDIT:
            return TPMS_COMMAND_AUDIT_INFO_Marshal(
                (TPMS_COMMAND_AUDIT_INFO*)&(source->commandAudit), buffer, size);
        case TPM_ST_ATTEST_SESSION_AUDIT:
            return TPMS_SESSION_AUDIT_INFO_Marshal(
                (TPMS_SESSION_AUDIT_INFO*)&(source->sessionAudit), buffer, size);
        case TPM_ST_ATTEST_TIME:
            return TPMS_TIME_ATTEST_INFO_Marshal(
```

```
                (TPMS_TIME_ATTEST_INFO*)&(source->time), buffer, size);
        case TPM_ST_ATTEST_NV:
            return TPMS_NV_CERTIFY_INFO_Marshal(
                (TPMS_NV_CERTIFY_INFO*)&(source->nv), buffer, size);
        case TPM_ST_ATTEST_NV_DIGEST:
            return TPMS_NV_DIGEST_CERTIFY_INFO_Marshal(
                (TPMS_NV_DIGEST_CERTIFY_INFO*)&(source->nvDigest), buffer, size);
    }
    return 0;
}

// Table "Definition of TPMS_ATTEST Structure" (Part 2: Structures)
UINT16
TPMS_ATTEST_Marshal(TPMS_ATTEST* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPM_CONSTANTS32_Marshal(
                          (TPM_CONSTANTS32*)&(source->magic), buffer, size));
    result        = (UINT16)(result
                      + TPMI_ST_ATTEST_Marshal(
                          (TPMI_ST_ATTEST*)&(source->type), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_NAME_Marshal(
                          (TPM2B_NAME*)&(source->qualifiedSigner), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_DATA_Marshal(
                          (TPM2B_DATA*)&(source->extraData), buffer, size));
    result        = (UINT16)(result
                      + TPMS_CLOCK_INFO_Marshal(
                          (TPMS_CLOCK_INFO*)&(source->clockInfo), buffer, size));
    result =
        (UINT16)(result
                 + UINT64_Marshal((UINT64*)&(source->firmwareVersion), buffer, size));
    result = (UINT16)(result
                      + TPMU_ATTEST_Marshal((TPMU_ATTEST*)&(source->attested),
                                            buffer,
                                            size,
                                            (UINT32)source->type));
    return result;
}

// Table "Definition of TPM2B_ATTEST Structure" (Part 2: Structures)
UINT16
TPM2B_ATTEST_Marshal(TPM2B_ATTEST* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.attestationData),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_AUTH_COMMAND Structure" (Part 2: Structures)
TPM_RC
TPMS_AUTH_COMMAND_Unmarshal(TPMS_AUTH_COMMAND* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPMI_SH_AUTH_SESSION_Unmarshal(
```

```
        (TPMI_SH_AUTH_SESSION*)&(target->sessionHandle), buffer, size, 1);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_NONCE_Unmarshal((TPM2B_NONCE*)&(target->nonce), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMA_SESSION_Unmarshal(
            (TPMA_SESSION*)&(target->sessionAttributes), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_AUTH_Unmarshal((TPM2B_AUTH*)&(target->hmac), buffer, size);
    return result;
}

// Table "Definition of TPMS_AUTH_RESPONSE Structure" (Part 2: Structures)
UINT16
TPMS_AUTH_RESPONSE_Marshal(TPMS_AUTH_RESPONSE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM2B_NONCE_Marshal((TPM2B_NONCE*)&(source->nonce), buffer, size));
    result = (UINT16)(result
                      + TPMA_SESSION_Marshal(
                          (TPMA_SESSION*)&(source->sessionAttributes), buffer, size));
    result =
        (UINT16)(result
                 + TPM2B_AUTH_Marshal((TPM2B_AUTH*)&(source->hmac), buffer, size));
    return result;
}

// Table "Definition of TPMI_AES_KEY_BITS Type" (Part 2: Structures)
#  if ALG_AES
TPM_RC
TPMI_AES_KEY_BITS_Unmarshal(TPMI_AES_KEY_BITS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_KEY_BITS_Unmarshal((TPM_KEY_BITS*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#    if AES_128
            case 128:
#    endif  // AES_128
#    if AES_192
            case 192:
#    endif  // AES_192
#    if AES_256
            case 256:
#    endif  // AES_256
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#    if !USE_MARSHALING_DEFINES
UINT16
TPMI_AES_KEY_BITS_Marshal(TPMI_AES_KEY_BITS* source, BYTE** buffer, INT32* size)
{
    return TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)source, buffer, size);
}
#    endif  // !USE_MARSHALING_DEFINES
#  endif    // ALG_AES

// Table "Definition of TPMI_SM4_KEY_BITS Type" (Part 2: Structures)
```

```
#  if ALG_SM4
TPM_RC
TPMI_SM4_KEY_BITS_Unmarshal(TPMI_SM4_KEY_BITS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_KEY_BITS_Unmarshal((TPM_KEY_BITS*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#    if SM4_128
            case 128:
#    endif  // SM4_128
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#    if !USE_MARSHALING_DEFINES
UINT16
TPMI_SM4_KEY_BITS_Marshal(TPMI_SM4_KEY_BITS* source, BYTE** buffer, INT32* size)
{
    return TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)source, buffer, size);
}
#    endif  // !USE_MARSHALING_DEFINES
#  endif    // ALG_SM4

// Table "Definition of TPMI_CAMELLIA_KEY_BITS Type" (Part 2: Structures)
#  if ALG_CAMELLIA
TPM_RC
TPMI_CAMELLIA_KEY_BITS_Unmarshal(
    TPMI_CAMELLIA_KEY_BITS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_KEY_BITS_Unmarshal((TPM_KEY_BITS*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#    if CAMELLIA_128
            case 128:
#    endif  // CAMELLIA_128
#    if CAMELLIA_192
            case 192:
#    endif  // CAMELLIA_192
#    if CAMELLIA_256
            case 256:
#    endif  // CAMELLIA_256
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#    if !USE_MARSHALING_DEFINES
UINT16
TPMI_CAMELLIA_KEY_BITS_Marshal(
    TPMI_CAMELLIA_KEY_BITS* source, BYTE** buffer, INT32* size)
{
    return TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)source, buffer, size);
}
```

```
#    endif  // !USE_MARSHALING_DEFINES
#  endif    // ALG_CAMELLIA

// Table "Definition of TPMU_SYM_KEY_BITS Union" (Part 2: Structures)
TPM_RC
TPMU_SYM_KEY_BITS_Unmarshal(
    TPMU_SYM_KEY_BITS* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_AES
        case TPM_ALG_AES:
            return TPMI_AES_KEY_BITS_Unmarshal(
                (TPMI_AES_KEY_BITS*)&(target->aes), buffer, size);
#  endif  // ALG_AES
#  if ALG_SM4
        case TPM_ALG_SM4:
            return TPMI_SM4_KEY_BITS_Unmarshal(
                (TPMI_SM4_KEY_BITS*)&(target->sm4), buffer, size);
#  endif  // ALG_SM4
#  if ALG_CAMELLIA
        case TPM_ALG_CAMELLIA:
            return TPMI_CAMELLIA_KEY_BITS_Unmarshal(
                (TPMI_CAMELLIA_KEY_BITS*)&(target->camellia), buffer, size);
#  endif  // ALG_CAMELLIA
#  if ALG_XOR
        case TPM_ALG_XOR:
            return TPMI_ALG_HASH_Unmarshal(
                (TPMI_ALG_HASH*)&(target->xor), buffer, size, 0);
#  endif  // ALG_XOR
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_SYM_KEY_BITS_Marshal(
    TPMU_SYM_KEY_BITS* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_AES
        case TPM_ALG_AES:
            return TPMI_AES_KEY_BITS_Marshal(
                (TPMI_AES_KEY_BITS*)&(source->aes), buffer, size);
#  endif  // ALG_AES
#  if ALG_SM4
        case TPM_ALG_SM4:
            return TPMI_SM4_KEY_BITS_Marshal(
                (TPMI_SM4_KEY_BITS*)&(source->sm4), buffer, size);
#  endif  // ALG_SM4
#  if ALG_CAMELLIA
        case TPM_ALG_CAMELLIA:
            return TPMI_CAMELLIA_KEY_BITS_Marshal(
                (TPMI_CAMELLIA_KEY_BITS*)&(source->camellia), buffer, size);
#  endif  // ALG_CAMELLIA
#  if ALG_XOR
        case TPM_ALG_XOR:
            return TPMI_ALG_HASH_Marshal(
                (TPMI_ALG_HASH*)&(source->xor), buffer, size);
#  endif  // ALG_XOR
    }
    return 0;
}

// Table "Definition of TPMU_SYM_MODE Union" (Part 2: Structures)
```

```
TPM_RC
TPMU_SYM_MODE_Unmarshal(
    TPMU_SYM_MODE* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_AES
        case TPM_ALG_AES:
            return TPMI_ALG_SYM_MODE_Unmarshal(
                (TPMI_ALG_SYM_MODE*)&(target->aes), buffer, size, 1);
#   endif  // ALG_AES
#   if ALG_SM4
        case TPM_ALG_SM4:
            return TPMI_ALG_SYM_MODE_Unmarshal(
                (TPMI_ALG_SYM_MODE*)&(target->sm4), buffer, size, 1);
#   endif  // ALG_SM4
#   if ALG_CAMELLIA
        case TPM_ALG_CAMELLIA:
            return TPMI_ALG_SYM_MODE_Unmarshal(
                (TPMI_ALG_SYM_MODE*)&(target->camellia), buffer, size, 1);
#   endif  // ALG_CAMELLIA
#   if ALG_XOR
        case TPM_ALG_XOR:
            return TPM_RC_SUCCESS;
#   endif  // ALG_XOR
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_SYM_MODE_Marshal(
    TPMU_SYM_MODE* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_AES
        case TPM_ALG_AES:
            return TPMI_ALG_SYM_MODE_Marshal(
                (TPMI_ALG_SYM_MODE*)&(source->aes), buffer, size);
#   endif  // ALG_AES
#   if ALG_SM4
        case TPM_ALG_SM4:
            return TPMI_ALG_SYM_MODE_Marshal(
                (TPMI_ALG_SYM_MODE*)&(source->sm4), buffer, size);
#   endif  // ALG_SM4
#   if ALG_CAMELLIA
        case TPM_ALG_CAMELLIA:
            return TPMI_ALG_SYM_MODE_Marshal(
                (TPMI_ALG_SYM_MODE*)&(source->camellia), buffer, size);
#   endif  // ALG_CAMELLIA
    }
    return 0;
}

// Table "Definition of TPMT_SYM_DEF Structure" (Part 2: Structures)
TPM_RC
TPMT_SYM_DEF_Unmarshal(TPMT_SYM_DEF* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_SYM_Unmarshal(
        (TPMI_ALG_SYM*)&(target->algorithm), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SYM_KEY_BITS_Unmarshal((TPMU_SYM_KEY_BITS*)&(target->keyBits),
                                             buffer,
                                             size,
```

```
                                            (UINT32)target->algorithm);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SYM_MODE_Unmarshal(
            (TPMU_SYM_MODE*)&(target->mode), buffer, size, (UINT32)target->algorithm);
    return result;
}
UINT16
TPMT_SYM_DEF_Marshal(TPMT_SYM_DEF* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_SYM_Marshal(
                          (TPMI_ALG_SYM*)&(source->algorithm), buffer, size));
    result =
        (UINT16)(result
                 + TPMU_SYM_KEY_BITS_Marshal((TPMU_SYM_KEY_BITS*)&(source->keyBits),
                                             buffer,
                                             size,
                                             (UINT32)source->algorithm));
    result = (UINT16)(result
                      + TPMU_SYM_MODE_Marshal((TPMU_SYM_MODE*)&(source->mode),
                                              buffer,
                                              size,
                                              (UINT32)source->algorithm));
    return result;
}

// Table "Definition of TPMT_SYM_DEF_OBJECT Structure" (Part 2: Structures)
TPM_RC
TPMT_SYM_DEF_OBJECT_Unmarshal(
    TPMT_SYM_DEF_OBJECT* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_SYM_OBJECT_Unmarshal(
        (TPMI_ALG_SYM_OBJECT*)&(target->algorithm), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SYM_KEY_BITS_Unmarshal((TPMU_SYM_KEY_BITS*)&(target->keyBits),
                                             buffer,
                                             size,
                                             (UINT32)target->algorithm);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SYM_MODE_Unmarshal(
            (TPMU_SYM_MODE*)&(target->mode), buffer, size, (UINT32)target->algorithm);
    return result;
}
UINT16
TPMT_SYM_DEF_OBJECT_Marshal(TPMT_SYM_DEF_OBJECT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_SYM_OBJECT_Marshal(
                          (TPMI_ALG_SYM_OBJECT*)&(source->algorithm), buffer, size));
    result =
        (UINT16)(result
                 + TPMU_SYM_KEY_BITS_Marshal((TPMU_SYM_KEY_BITS*)&(source->keyBits),
                                             buffer,
                                             size,
                                             (UINT32)source->algorithm));
    result = (UINT16)(result
                      + TPMU_SYM_MODE_Marshal((TPMU_SYM_MODE*)&(source->mode),
                                              buffer,
                                              size,
                                              (UINT32)source->algorithm));
    return result;
}
```

```
// Table "Definition of TPM2B_SYM_KEY Structure" (Part 2: Structures)
TPM_RC
TPM2B_SYM_KEY_Unmarshal(TPM2B_SYM_KEY* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > MAX_SYM_KEY_BYTES))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_SYM_KEY_Marshal(TPM2B_SYM_KEY* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_SYMCIPHER_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_SYMCIPHER_PARMS_Unmarshal(
    TPMS_SYMCIPHER_PARMS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPMT_SYM_DEF_OBJECT_Unmarshal(
        (TPMT_SYM_DEF_OBJECT*)&(target->sym), buffer, size, 0);
    return result;
}
UINT16
TPMS_SYMCIPHER_PARMS_Marshal(TPMS_SYMCIPHER_PARMS* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMT_SYM_DEF_OBJECT_Marshal(
                          (TPMT_SYM_DEF_OBJECT*)&(source->sym), buffer, size));
    return result;
}

// Table "Definition of TPM2B_LABEL Structure" (Part 2: Structures)
TPM_RC
TPM2B_LABEL_Unmarshal(TPM2B_LABEL* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > LABEL_MAX_BUFFER))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_LABEL_Marshal(TPM2B_LABEL* source, BYTE** buffer, INT32* size)
{
```

```
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_DERIVE Structure" (Part 2: Structures)
TPM_RC
TPMS_DERIVE_Unmarshal(TPMS_DERIVE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM2B_LABEL_Unmarshal((TPM2B_LABEL*)&(target->label), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPM2B_LABEL_Unmarshal((TPM2B_LABEL*)&(target->context), buffer, size);
    return result;
}
UINT16
TPMS_DERIVE_Marshal(TPMS_DERIVE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPM2B_LABEL_Marshal((TPM2B_LABEL*)&(source->label), buffer, size));
    result = (UINT16)(result
                      + TPM2B_LABEL_Marshal(
                          (TPM2B_LABEL*)&(source->context), buffer, size));
    return result;
}

// Table "Definition of TPM2B_DERIVE Structure" (Part 2: Structures)
TPM_RC
TPM2B_DERIVE_Unmarshal(TPM2B_DERIVE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMS_DERIVE)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_DERIVE_Marshal(TPM2B_DERIVE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
```

```
}

// Table "Definition of TPM2B_SENSITIVE_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_SENSITIVE_DATA_Unmarshal(
    TPM2B_SENSITIVE_DATA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMU_SENSITIVE_CREATE)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_SENSITIVE_DATA_Marshal(TPM2B_SENSITIVE_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_SENSITIVE_CREATE Structure" (Part 2: Structures)
TPM_RC
TPMS_SENSITIVE_CREATE_Unmarshal(
    TPMS_SENSITIVE_CREATE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM2B_AUTH_Unmarshal((TPM2B_AUTH*)&(target->userAuth), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_SENSITIVE_DATA_Unmarshal(
            (TPM2B_SENSITIVE_DATA*)&(target->data), buffer, size);
    return result;
}

// Table "Definition of TPM2B_SENSITIVE_CREATE Structure" (Part 2: Structures)
TPM_RC
TPM2B_SENSITIVE_CREATE_Unmarshal(
    TPM2B_SENSITIVE_CREATE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->size), buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // if size is zero, then the required structure is missing
        if(target->size == 0)
            result = TPM_RC_SIZE;
        else
        {
            INT32 startSize = *size;
            result          = TPMS_SENSITIVE_CREATE_Unmarshal(
                (TPMS_SENSITIVE_CREATE*)&(target->sensitive), buffer, size);
            if((result == TPM_RC_SUCCESS) && (target->size != (startSize - *size)))
                result = TPM_RC_SIZE;
        }
```

```
    }
    return result;
}

// Table "Definition of TPMS_SCHEME_HASH Structure" (Part 2: Structures)
TPM_RC
TPMS_SCHEME_HASH_Unmarshal(TPMS_SCHEME_HASH* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result =
        TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH*)&(target->hashAlg), buffer, size, 0);
    return result;
}
UINT16
TPMS_SCHEME_HASH_Marshal(TPMS_SCHEME_HASH* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                    + TPMI_ALG_HASH_Marshal(
                        (TPMI_ALG_HASH*)&(source->hashAlg), buffer, size));
    return result;
}

// Table "Definition of TPMS_SCHEME_ECDAA Structure" (Part 2: Structures)
#  if ALG_ECC
TPM_RC
TPMS_SCHEME_ECDAA_Unmarshal(TPMS_SCHEME_ECDAA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result =
        TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH*)&(target->hashAlg), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
        result = UINT16_Unmarshal((UINT16*)&(target->count), buffer, size);
    return result;
}
UINT16
TPMS_SCHEME_ECDAA_Marshal(TPMS_SCHEME_ECDAA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                    + TPMI_ALG_HASH_Marshal(
                        (TPMI_ALG_HASH*)&(source->hashAlg), buffer, size));
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->count), buffer, size));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of TPMI_ALG_KEYEDHASH_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_KEYEDHASH_SCHEME_Unmarshal(
    TPMI_ALG_KEYEDHASH_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_HMAC
            case TPM_ALG_HMAC:
#  endif  // ALG_HMAC
#  if ALG_XOR
            case TPM_ALG_XOR:
#  endif  // ALG_XOR
                break;
```

```
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_KEYEDHASH_SCHEME_Marshal(
    TPMI_ALG_KEYEDHASH_SCHEME* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for HMAC_SIG_SCHEME" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SCHEME_HMAC_Unmarshal(TPMS_SCHEME_HMAC* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SCHEME_HMAC_Marshal(TPMS_SCHEME_HMAC* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_SCHEME_XOR Structure" (Part 2: Structures)
TPM_RC
TPMS_SCHEME_XOR_Unmarshal(TPMS_SCHEME_XOR* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result =
        TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH*)&(target->hashAlg), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
        result =
            TPMI_ALG_KDF_Unmarshal((TPMI_ALG_KDF*)&(target->kdf), buffer, size, 1);
    return result;
}
UINT16
TPMS_SCHEME_XOR_Marshal(TPMS_SCHEME_XOR* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
                          (TPMI_ALG_HASH*)&(source->hashAlg), buffer, size));
    result =
        (UINT16)(result
                 + TPMI_ALG_KDF_Marshal((TPMI_ALG_KDF*)&(source->kdf), buffer, size));
    return result;
}

// Table "Definition of TPMU_SCHEME_KEYEDHASH Union" (Part 2: Structures)
TPM_RC
TPMU_SCHEME_KEYEDHASH_Unmarshal(
    TPMU_SCHEME_KEYEDHASH* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_HMAC
        case TPM_ALG_HMAC:
            return TPMS_SCHEME_HMAC_Unmarshal(
```

```
                (TPMS_SCHEME_HMAC*)&(target->hmac), buffer, size);
#   endif  // ALG_HMAC
#   if ALG_XOR
        case TPM_ALG_XOR:
            return TPMS_SCHEME_XOR_Unmarshal(
                (TPMS_SCHEME_XOR*)&(target->xor), buffer, size);
#   endif  // ALG_XOR
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_SCHEME_KEYEDHASH_Marshal(
    TPMU_SCHEME_KEYEDHASH* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_HMAC
        case TPM_ALG_HMAC:
            return TPMS_SCHEME_HMAC_Marshal(
                (TPMS_SCHEME_HMAC*)&(source->hmac), buffer, size);
#   endif  // ALG_HMAC
#   if ALG_XOR
        case TPM_ALG_XOR:
            return TPMS_SCHEME_XOR_Marshal(
                (TPMS_SCHEME_XOR*)&(source->xor), buffer, size);
#   endif  // ALG_XOR
    }
    return 0;
}

// Table "Definition of TPMT_KEYEDHASH_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_KEYEDHASH_SCHEME_Unmarshal(
    TPMT_KEYEDHASH_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_KEYEDHASH_SCHEME_Unmarshal(
        (TPMI_ALG_KEYEDHASH_SCHEME*)&(target->scheme), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SCHEME_KEYEDHASH_Unmarshal(
            (TPMU_SCHEME_KEYEDHASH*)&(target->details),
            buffer,
            size,
            (UINT32)target->scheme);
    return result;
}
UINT16
TPMT_KEYEDHASH_SCHEME_Marshal(
    TPMT_KEYEDHASH_SCHEME* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPMI_ALG_KEYEDHASH_SCHEME_Marshal(
                     (TPMI_ALG_KEYEDHASH_SCHEME*)&(source->scheme), buffer, size));
    result = (UINT16)(result
                      + TPMU_SCHEME_KEYEDHASH_Marshal(
                          (TPMU_SCHEME_KEYEDHASH*)&(source->details),
                          buffer,
                          size,
                          (UINT32)source->scheme));
    return result;
}
```

```
// Table "Definition of Types for RSA Signature Schemes" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_RSASSA_Unmarshal(
    TPMS_SIG_SCHEME_RSASSA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_RSASSA_Marshal(
    TPMS_SIG_SCHEME_RSASSA* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_SIG_SCHEME_RSAPSS_Unmarshal(
    TPMS_SIG_SCHEME_RSAPSS* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_RSAPSS_Marshal(
    TPMS_SIG_SCHEME_RSAPSS* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for ECC Signature Schemes" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIG_SCHEME_ECDSA_Unmarshal(
    TPMS_SIG_SCHEME_ECDSA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_ECDSA_Marshal(
    TPMS_SIG_SCHEME_ECDSA* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_SIG_SCHEME_ECDAA_Unmarshal(
    TPMS_SIG_SCHEME_ECDAA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_ECDAA_Unmarshal((TPMS_SCHEME_ECDAA*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_ECDAA_Marshal(
    TPMS_SIG_SCHEME_ECDAA* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_ECDAA_Marshal((TPMS_SCHEME_ECDAA*)source, buffer, size);
}
TPM_RC
TPMS_SIG_SCHEME_SM2_Unmarshal(TPMS_SIG_SCHEME_SM2* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_SM2_Marshal(TPMS_SIG_SCHEME_SM2* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_SIG_SCHEME_ECSCHNORR_Unmarshal(
```

```
    TPMS_SIG_SCHEME_ECSCHNORR* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_ECSCHNORR_Marshal(
    TPMS_SIG_SCHEME_ECSCHNORR* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_SIG_SCHEME_EDDSA_Unmarshal(
    TPMS_SIG_SCHEME_EDDSA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_EDDSA_Marshal(
    TPMS_SIG_SCHEME_EDDSA* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_SIG_SCHEME_EDDSA_PH_Unmarshal(
    TPMS_SIG_SCHEME_EDDSA_PH* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_SIG_SCHEME_EDDSA_PH_Marshal(
    TPMS_SIG_SCHEME_EDDSA_PH* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_SIG_SCHEME Union" (Part 2: Structures)
TPM_RC
TPMU_SIG_SCHEME_Unmarshal(
    TPMU_SIG_SCHEME* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_HMAC
        case TPM_ALG_HMAC:
            return TPMS_SCHEME_HMAC_Unmarshal(
                (TPMS_SCHEME_HMAC*)&(target->hmac), buffer, size);
#  endif  // ALG_HMAC
#  if ALG_RSASSA
        case TPM_ALG_RSASSA:
            return TPMS_SIG_SCHEME_RSASSA_Unmarshal(
                (TPMS_SIG_SCHEME_RSASSA*)&(target->rsassa), buffer, size);
#  endif  // ALG_RSASSA
#  if ALG_RSAPSS
        case TPM_ALG_RSAPSS:
            return TPMS_SIG_SCHEME_RSAPSS_Unmarshal(
                (TPMS_SIG_SCHEME_RSAPSS*)&(target->rsapss), buffer, size);
#  endif  // ALG_RSAPSS
#  if ALG_ECDSA
        case TPM_ALG_ECDSA:
            return TPMS_SIG_SCHEME_ECDSA_Unmarshal(
                (TPMS_SIG_SCHEME_ECDSA*)&(target->ecdsa), buffer, size);
#  endif  // ALG_ECDSA
#  if ALG_ECDAA
        case TPM_ALG_ECDAA:
            return TPMS_SIG_SCHEME_ECDAA_Unmarshal(
```

```
                (TPMS_SIG_SCHEME_ECDAA*)&(target->ecdaa), buffer, size);
#   endif  // ALG_ECDAA
#   if ALG_SM2
        case TPM_ALG_SM2:
            return TPMS_SIG_SCHEME_SM2_Unmarshal(
                (TPMS_SIG_SCHEME_SM2*)&(target->sm2), buffer, size);
#   endif  // ALG_SM2
#   if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            return TPMS_SIG_SCHEME_ECSCHNORR_Unmarshal(
                (TPMS_SIG_SCHEME_ECSCHNORR*)&(target->ecschnorr), buffer, size);
#   endif  // ALG_ECSCHNORR
#   if ALG_EDDSA
        case TPM_ALG_EDDSA:
            return TPMS_SIG_SCHEME_EDDSA_Unmarshal(
                (TPMS_SIG_SCHEME_EDDSA*)&(target->eddsa), buffer, size);
#   endif  // ALG_EDDSA
#   if ALG_EDDSA_PH
        case TPM_ALG_EDDSA_PH:
            return TPMS_SIG_SCHEME_EDDSA_PH_Unmarshal(
                (TPMS_SIG_SCHEME_EDDSA_PH*)&(target->eddsa_ph), buffer, size);
#   endif  // ALG_EDDSA_PH
#   if ALG_LMS
        case TPM_ALG_LMS:
            return TPMS_SIG_SCHEME_LMS_Unmarshal(
                (TPMS_SIG_SCHEME_LMS*)&(target->lms), buffer, size);
#   endif  // ALG_LMS
#   if ALG_XMSS
        case TPM_ALG_XMSS:
            return TPMS_SIG_SCHEME_XMSS_Unmarshal(
                (TPMS_SIG_SCHEME_XMSS*)&(target->xmss), buffer, size);
#   endif  // ALG_XMSS
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_SIG_SCHEME_Marshal(
    TPMU_SIG_SCHEME* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_HMAC
        case TPM_ALG_HMAC:
            return TPMS_SCHEME_HMAC_Marshal(
                (TPMS_SCHEME_HMAC*)&(source->hmac), buffer, size);
#   endif  // ALG_HMAC
#   if ALG_RSASSA
        case TPM_ALG_RSASSA:
            return TPMS_SIG_SCHEME_RSASSA_Marshal(
                (TPMS_SIG_SCHEME_RSASSA*)&(source->rsassa), buffer, size);
#   endif  // ALG_RSASSA
#   if ALG_RSAPSS
        case TPM_ALG_RSAPSS:
            return TPMS_SIG_SCHEME_RSAPSS_Marshal(
                (TPMS_SIG_SCHEME_RSAPSS*)&(source->rsapss), buffer, size);
#   endif  // ALG_RSAPSS
#   if ALG_ECDSA
        case TPM_ALG_ECDSA:
            return TPMS_SIG_SCHEME_ECDSA_Marshal(
                (TPMS_SIG_SCHEME_ECDSA*)&(source->ecdsa), buffer, size);
#   endif  // ALG_ECDSA
#   if ALG_ECDAA
        case TPM_ALG_ECDAA:
            return TPMS_SIG_SCHEME_ECDAA_Marshal(
```

```
                (TPMS_SIG_SCHEME_ECDAA*)&(source->ecdaa), buffer, size);
#   endif  // ALG_ECDAA
#   if ALG_SM2
        case TPM_ALG_SM2:
            return TPMS_SIG_SCHEME_SM2_Marshal(
                (TPMS_SIG_SCHEME_SM2*)&(source->sm2), buffer, size);
#   endif  // ALG_SM2
#   if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            return TPMS_SIG_SCHEME_ECSCHNORR_Marshal(
                (TPMS_SIG_SCHEME_ECSCHNORR*)&(source->ecschnorr), buffer, size);
#   endif  // ALG_ECSCHNORR
#   if ALG_EDDSA
        case TPM_ALG_EDDSA:
            return TPMS_SIG_SCHEME_EDDSA_Marshal(
                (TPMS_SIG_SCHEME_EDDSA*)&(source->eddsa), buffer, size);
#   endif  // ALG_EDDSA
#   if ALG_EDDSA_PH
        case TPM_ALG_EDDSA_PH:
            return TPMS_SIG_SCHEME_EDDSA_PH_Marshal(
                (TPMS_SIG_SCHEME_EDDSA_PH*)&(source->eddsa_ph), buffer, size);
#   endif  // ALG_EDDSA_PH
#   if ALG_LMS
        case TPM_ALG_LMS:
            return TPMS_SIG_SCHEME_LMS_Marshal(
                (TPMS_SIG_SCHEME_LMS*)&(source->lms), buffer, size);
#   endif  // ALG_LMS
#   if ALG_XMSS
        case TPM_ALG_XMSS:
            return TPMS_SIG_SCHEME_XMSS_Marshal(
                (TPMS_SIG_SCHEME_XMSS*)&(source->xmss), buffer, size);
#   endif  // ALG_XMSS
    }
    return 0;
}

// Table "Definition of TPMT_SIG_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_SIG_SCHEME_Unmarshal(
    TPMT_SIG_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_SIG_SCHEME_Unmarshal(
        (TPMI_ALG_SIG_SCHEME*)&(target->scheme), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SIG_SCHEME_Unmarshal((TPMU_SIG_SCHEME*)&(target->details),
                                           buffer,
                                           size,
                                           (UINT32)target->scheme);
    return result;
}
UINT16
TPMT_SIG_SCHEME_Marshal(TPMT_SIG_SCHEME* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_SIG_SCHEME_Marshal(
                          (TPMI_ALG_SIG_SCHEME*)&(source->scheme), buffer, size));
    result        = (UINT16)(result
                      + TPMU_SIG_SCHEME_Marshal((TPMU_SIG_SCHEME*)&(source->details),
                                                buffer,
                                                size,
                                                (UINT32)source->scheme));
    return result;
}
```

```
// Table "Definition of Types for Encryption Schemes" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_ENC_SCHEME_RSAES_Unmarshal(
    TPMS_ENC_SCHEME_RSAES* target, BYTE** buffer, INT32* size)
{
    return TPMS_EMPTY_Unmarshal((TPMS_EMPTY*)target, buffer, size);
}
UINT16
TPMS_ENC_SCHEME_RSAES_Marshal(
    TPMS_ENC_SCHEME_RSAES* source, BYTE** buffer, INT32* size)
{
    return TPMS_EMPTY_Marshal((TPMS_EMPTY*)source, buffer, size);
}
TPM_RC
TPMS_ENC_SCHEME_OAEP_Unmarshal(
    TPMS_ENC_SCHEME_OAEP* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_ENC_SCHEME_OAEP_Marshal(TPMS_ENC_SCHEME_OAEP* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for ECC Key Exchange" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_KEY_SCHEME_ECDH_Unmarshal(
    TPMS_KEY_SCHEME_ECDH* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_KEY_SCHEME_ECDH_Marshal(TPMS_KEY_SCHEME_ECDH* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_KEY_SCHEME_SM2_Unmarshal(TPMS_KEY_SCHEME_SM2* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_KEY_SCHEME_SM2_Marshal(TPMS_KEY_SCHEME_SM2* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_KEY_SCHEME_ECMQV_Unmarshal(
    TPMS_KEY_SCHEME_ECMQV* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_KEY_SCHEME_ECMQV_Marshal(
    TPMS_KEY_SCHEME_ECMQV* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of Types for KDF Schemes" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
```

```
TPM_RC
TPMS_KDF_SCHEME_MGF1_Unmarshal(
    TPMS_KDF_SCHEME_MGF1* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_KDF_SCHEME_MGF1_Marshal(TPMS_KDF_SCHEME_MGF1* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_KDF_SCHEME_KDF1_SP800_56A_Unmarshal(
    TPMS_KDF_SCHEME_KDF1_SP800_56A* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_KDF_SCHEME_KDF1_SP800_56A_Marshal(
    TPMS_KDF_SCHEME_KDF1_SP800_56A* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_KDF_SCHEME_KDF2_Unmarshal(
    TPMS_KDF_SCHEME_KDF2* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_KDF_SCHEME_KDF2_Marshal(TPMS_KDF_SCHEME_KDF2* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
TPM_RC
TPMS_KDF_SCHEME_KDF1_SP800_108_Unmarshal(
    TPMS_KDF_SCHEME_KDF1_SP800_108* target, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Unmarshal((TPMS_SCHEME_HASH*)target, buffer, size);
}
UINT16
TPMS_KDF_SCHEME_KDF1_SP800_108_Marshal(
    TPMS_KDF_SCHEME_KDF1_SP800_108* source, BYTE** buffer, INT32* size)
{
    return TPMS_SCHEME_HASH_Marshal((TPMS_SCHEME_HASH*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_KDF_SCHEME Union" (Part 2: Structures)
TPM_RC
TPMU_KDF_SCHEME_Unmarshal(
    TPMU_KDF_SCHEME* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_MGF1
        case TPM_ALG_MGF1:
            return TPMS_KDF_SCHEME_MGF1_Unmarshal(
                (TPMS_KDF_SCHEME_MGF1*)&(target->mgf1), buffer, size);
#  endif  // ALG_MGF1
#  if ALG_KDF1_SP800_56A
        case TPM_ALG_KDF1_SP800_56A:
            return TPMS_KDF_SCHEME_KDF1_SP800_56A_Unmarshal(
                (TPMS_KDF_SCHEME_KDF1_SP800_56A*)&(target->kdf1_sp800_56a),
                buffer,
                size);
```

```
#  endif  // ALG_KDF1_SP800_56A
#  if ALG_KDF2
        case TPM_ALG_KDF2:
            return TPMS_KDF_SCHEME_KDF2_Unmarshal(
                (TPMS_KDF_SCHEME_KDF2*)&(target->kdf2), buffer, size);
#  endif  // ALG_KDF2
#  if ALG_KDF1_SP800_108
        case TPM_ALG_KDF1_SP800_108:
            return TPMS_KDF_SCHEME_KDF1_SP800_108_Unmarshal(
                (TPMS_KDF_SCHEME_KDF1_SP800_108*)&(target->kdf1_sp800_108),
                buffer,
                size);
#  endif  // ALG_KDF1_SP800_108
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_KDF_SCHEME_Marshal(
    TPMU_KDF_SCHEME* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_MGF1
        case TPM_ALG_MGF1:
            return TPMS_KDF_SCHEME_MGF1_Marshal(
                (TPMS_KDF_SCHEME_MGF1*)&(source->mgf1), buffer, size);
#  endif  // ALG_MGF1
#  if ALG_KDF1_SP800_56A
        case TPM_ALG_KDF1_SP800_56A:
            return TPMS_KDF_SCHEME_KDF1_SP800_56A_Marshal(
                (TPMS_KDF_SCHEME_KDF1_SP800_56A*)&(source->kdf1_sp800_56a),
                buffer,
                size);
#  endif  // ALG_KDF1_SP800_56A
#  if ALG_KDF2
        case TPM_ALG_KDF2:
            return TPMS_KDF_SCHEME_KDF2_Marshal(
                (TPMS_KDF_SCHEME_KDF2*)&(source->kdf2), buffer, size);
#  endif  // ALG_KDF2
#  if ALG_KDF1_SP800_108
        case TPM_ALG_KDF1_SP800_108:
            return TPMS_KDF_SCHEME_KDF1_SP800_108_Marshal(
                (TPMS_KDF_SCHEME_KDF1_SP800_108*)&(source->kdf1_sp800_108),
                buffer,
                size);
#  endif  // ALG_KDF1_SP800_108
    }
    return 0;
}

// Table "Definition of TPMT_KDF_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_KDF_SCHEME_Unmarshal(
    TPMT_KDF_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result =
        TPMI_ALG_KDF_Unmarshal((TPMI_ALG_KDF*)&(target->scheme), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_KDF_SCHEME_Unmarshal((TPMU_KDF_SCHEME*)&(target->details),
                                           buffer,
                                           size,
                                           (UINT32)target->scheme);
    return result;
```

```
}
UINT16
TPMT_KDF_SCHEME_Marshal(TPMT_KDF_SCHEME* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_KDF_Marshal(
                          (TPMI_ALG_KDF*)&(source->scheme), buffer, size));
    result        = (UINT16)(result
                      + TPMU_KDF_SCHEME_Marshal((TPMU_KDF_SCHEME*)&(source->details),
                                                buffer,
                                                size,
                                                (UINT32)source->scheme));
    return result;
}

// Table "Definition of TPMI_ALG_ASYM_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_ASYM_SCHEME_Unmarshal(
    TPMI_ALG_ASYM_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_RSASSA
            case TPM_ALG_RSASSA:
#  endif  // ALG_RSASSA
#  if ALG_RSAES
            case TPM_ALG_RSAES:
#  endif  // ALG_RSAES
#  if ALG_RSAPSS
            case TPM_ALG_RSAPSS:
#  endif  // ALG_RSAPSS
#  if ALG_OAEP
            case TPM_ALG_OAEP:
#  endif  // ALG_OAEP
#  if ALG_ECDSA
            case TPM_ALG_ECDSA:
#  endif  // ALG_ECDSA
#  if ALG_ECDH
            case TPM_ALG_ECDH:
#  endif  // ALG_ECDH
#  if ALG_ECDAA
            case TPM_ALG_ECDAA:
#  endif  // ALG_ECDAA
#  if ALG_SM2
            case TPM_ALG_SM2:
#  endif  // ALG_SM2
#  if ALG_ECSCHNORR
            case TPM_ALG_ECSCHNORR:
#  endif  // ALG_ECSCHNORR
#  if ALG_ECMQV
            case TPM_ALG_ECMQV:
#  endif  // ALG_ECMQV
#  if ALG_EDDSA
            case TPM_ALG_EDDSA:
#  endif  // ALG_EDDSA
#  if ALG_EDDSA_PH
            case TPM_ALG_EDDSA_PH:
#  endif  // ALG_EDDSA_PH
#  if ALG_LMS
            case TPM_ALG_LMS:
#  endif  // ALG_LMS
```

```
#  if ALG_XMSS
            case TPM_ALG_XMSS:
#  endif  // ALG_XMSS
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_ASYM_SCHEME_Marshal(TPMI_ALG_ASYM_SCHEME* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_ASYM_SCHEME Union" (Part 2: Structures)
TPM_RC
TPMU_ASYM_SCHEME_Unmarshal(
    TPMU_ASYM_SCHEME* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_RSASSA
        case TPM_ALG_RSASSA:
            return TPMS_SIG_SCHEME_RSASSA_Unmarshal(
                (TPMS_SIG_SCHEME_RSASSA*)&(target->rsassa), buffer, size);
#  endif  // ALG_RSASSA
#  if ALG_RSAES
        case TPM_ALG_RSAES:
            return TPMS_ENC_SCHEME_RSAES_Unmarshal(
                (TPMS_ENC_SCHEME_RSAES*)&(target->rsaes), buffer, size);
#  endif  // ALG_RSAES
#  if ALG_RSAPSS
        case TPM_ALG_RSAPSS:
            return TPMS_SIG_SCHEME_RSAPSS_Unmarshal(
                (TPMS_SIG_SCHEME_RSAPSS*)&(target->rsapss), buffer, size);
#  endif  // ALG_RSAPSS
#  if ALG_OAEP
        case TPM_ALG_OAEP:
            return TPMS_ENC_SCHEME_OAEP_Unmarshal(
                (TPMS_ENC_SCHEME_OAEP*)&(target->oaep), buffer, size);
#  endif  // ALG_OAEP
#  if ALG_ECDSA
        case TPM_ALG_ECDSA:
            return TPMS_SIG_SCHEME_ECDSA_Unmarshal(
                (TPMS_SIG_SCHEME_ECDSA*)&(target->ecdsa), buffer, size);
#  endif  // ALG_ECDSA
#  if ALG_ECDH
        case TPM_ALG_ECDH:
            return TPMS_KEY_SCHEME_ECDH_Unmarshal(
                (TPMS_KEY_SCHEME_ECDH*)&(target->ecdh), buffer, size);
#  endif  // ALG_ECDH
#  if ALG_ECDAA
        case TPM_ALG_ECDAA:
            return TPMS_SIG_SCHEME_ECDAA_Unmarshal(
                (TPMS_SIG_SCHEME_ECDAA*)&(target->ecdaa), buffer, size);
#  endif  // ALG_ECDAA
#  if ALG_SM2
        case TPM_ALG_SM2:
            return TPMS_KEY_SCHEME_SM2_Unmarshal(
                (TPMS_KEY_SCHEME_SM2*)&(target->sm2), buffer, size);
```

```
#  endif  // ALG_SM2
#  if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            return TPMS_SIG_SCHEME_ECSCHNORR_Unmarshal(
                (TPMS_SIG_SCHEME_ECSCHNORR*)&(target->ecschnorr), buffer, size);
#  endif  // ALG_ECSCHNORR
#  if ALG_ECMQV
        case TPM_ALG_ECMQV:
            return TPMS_KEY_SCHEME_ECMQV_Unmarshal(
                (TPMS_KEY_SCHEME_ECMQV*)&(target->ecmqv), buffer, size);
#  endif  // ALG_ECMQV
#  if ALG_EDDSA
        case TPM_ALG_EDDSA:
            return TPMS_SIG_SCHEME_EDDSA_Unmarshal(
                (TPMS_SIG_SCHEME_EDDSA*)&(target->eddsa), buffer, size);
#  endif  // ALG_EDDSA
#  if ALG_EDDSA_PH
        case TPM_ALG_EDDSA_PH:
            return TPMS_SIG_SCHEME_EDDSA_PH_Unmarshal(
                (TPMS_SIG_SCHEME_EDDSA_PH*)&(target->eddsa_ph), buffer, size);
#  endif  // ALG_EDDSA_PH
#  if ALG_LMS
        case TPM_ALG_LMS:
            return TPMS_SIG_SCHEME_LMS_Unmarshal(
                (TPMS_SIG_SCHEME_LMS*)&(target->lms), buffer, size);
#  endif  // ALG_LMS
#  if ALG_XMSS
        case TPM_ALG_XMSS:
            return TPMS_SIG_SCHEME_XMSS_Unmarshal(
                (TPMS_SIG_SCHEME_XMSS*)&(target->xmss), buffer, size);
#  endif  // ALG_XMSS
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_ASYM_SCHEME_Marshal(
    TPMU_ASYM_SCHEME* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_RSASSA
        case TPM_ALG_RSASSA:
            return TPMS_SIG_SCHEME_RSASSA_Marshal(
                (TPMS_SIG_SCHEME_RSASSA*)&(source->rsassa), buffer, size);
#  endif  // ALG_RSASSA
#  if ALG_RSAES
        case TPM_ALG_RSAES:
            return TPMS_ENC_SCHEME_RSAES_Marshal(
                (TPMS_ENC_SCHEME_RSAES*)&(source->rsaes), buffer, size);
#  endif  // ALG_RSAES
#  if ALG_RSAPSS
        case TPM_ALG_RSAPSS:
            return TPMS_SIG_SCHEME_RSAPSS_Marshal(
                (TPMS_SIG_SCHEME_RSAPSS*)&(source->rsapss), buffer, size);
#  endif  // ALG_RSAPSS
#  if ALG_OAEP
        case TPM_ALG_OAEP:
            return TPMS_ENC_SCHEME_OAEP_Marshal(
                (TPMS_ENC_SCHEME_OAEP*)&(source->oaep), buffer, size);
#  endif  // ALG_OAEP
#  if ALG_ECDSA
        case TPM_ALG_ECDSA:
            return TPMS_SIG_SCHEME_ECDSA_Marshal(
                (TPMS_SIG_SCHEME_ECDSA*)&(source->ecdsa), buffer, size);
```

```
#  endif  // ALG_ECDSA
#  if ALG_ECDH
        case TPM_ALG_ECDH:
            return TPMS_KEY_SCHEME_ECDH_Marshal(
                (TPMS_KEY_SCHEME_ECDH*)&(source->ecdh), buffer, size);
#  endif  // ALG_ECDH
#  if ALG_ECDAA
        case TPM_ALG_ECDAA:
            return TPMS_SIG_SCHEME_ECDAA_Marshal(
                (TPMS_SIG_SCHEME_ECDAA*)&(source->ecdaa), buffer, size);
#  endif  // ALG_ECDAA
#  if ALG_SM2
        case TPM_ALG_SM2:
            return TPMS_KEY_SCHEME_SM2_Marshal(
                (TPMS_KEY_SCHEME_SM2*)&(source->sm2), buffer, size);
#  endif  // ALG_SM2
#  if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            return TPMS_SIG_SCHEME_ECSCHNORR_Marshal(
                (TPMS_SIG_SCHEME_ECSCHNORR*)&(source->ecschnorr), buffer, size);
#  endif  // ALG_ECSCHNORR
#  if ALG_ECMQV
        case TPM_ALG_ECMQV:
            return TPMS_KEY_SCHEME_ECMQV_Marshal(
                (TPMS_KEY_SCHEME_ECMQV*)&(source->ecmqv), buffer, size);
#  endif  // ALG_ECMQV
#  if ALG_EDDSA
        case TPM_ALG_EDDSA:
            return TPMS_SIG_SCHEME_EDDSA_Marshal(
                (TPMS_SIG_SCHEME_EDDSA*)&(source->eddsa), buffer, size);
#  endif  // ALG_EDDSA
#  if ALG_EDDSA_PH
        case TPM_ALG_EDDSA_PH:
            return TPMS_SIG_SCHEME_EDDSA_PH_Marshal(
                (TPMS_SIG_SCHEME_EDDSA_PH*)&(source->eddsa_ph), buffer, size);
#  endif  // ALG_EDDSA_PH
#  if ALG_LMS
        case TPM_ALG_LMS:
            return TPMS_SIG_SCHEME_LMS_Marshal(
                (TPMS_SIG_SCHEME_LMS*)&(source->lms), buffer, size);
#  endif  // ALG_LMS
#  if ALG_XMSS
        case TPM_ALG_XMSS:
            return TPMS_SIG_SCHEME_XMSS_Marshal(
                (TPMS_SIG_SCHEME_XMSS*)&(source->xmss), buffer, size);
#  endif  // ALG_XMSS
    }
    return 0;
}

// Table "Definition of TPMI_ALG_RSA_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_RSA_SCHEME_Unmarshal(
    TPMI_ALG_RSA_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_RSASSA
            case TPM_ALG_RSASSA:
#  endif  // ALG_RSASSA
#  if ALG_RSAES
            case TPM_ALG_RSAES:
```

```
#  endif  // ALG_RSAES
#  if ALG_RSAPSS
            case TPM_ALG_RSAPSS:
#  endif  // ALG_RSAPSS
#  if ALG_OAEP
            case TPM_ALG_OAEP:
#  endif  // ALG_OAEP
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_RSA_SCHEME_Marshal(TPMI_ALG_RSA_SCHEME* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMT_RSA_SCHEME Structure" (Part 2: Structures)
TPM_RC
TPMT_RSA_SCHEME_Unmarshal(
    TPMT_RSA_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_RSA_SCHEME_Unmarshal(
        (TPMI_ALG_RSA_SCHEME*)&(target->scheme), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_ASYM_SCHEME_Unmarshal((TPMU_ASYM_SCHEME*)&(target->details),
                                            buffer,
                                            size,
                                            (UINT32)target->scheme);
    return result;
}
UINT16
TPMT_RSA_SCHEME_Marshal(TPMT_RSA_SCHEME* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_RSA_SCHEME_Marshal(
                          (TPMI_ALG_RSA_SCHEME*)&(source->scheme), buffer, size));
    result =
        (UINT16)(result
                 + TPMU_ASYM_SCHEME_Marshal((TPMU_ASYM_SCHEME*)&(source->details),
                                            buffer,
                                            size,
                                            (UINT32)source->scheme));
    return result;
}

// Table "Definition of TPMI_ALG_RSA_DECRYPT Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_RSA_DECRYPT_Unmarshal(
    TPMI_ALG_RSA_DECRYPT* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
```

```
#  if ALG_RSAES
            case TPM_ALG_RSAES:
#  endif  // ALG_RSAES
#  if ALG_OAEP
            case TPM_ALG_OAEP:
#  endif  // ALG_OAEP
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_RSA_DECRYPT_Marshal(TPMI_ALG_RSA_DECRYPT* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMT_RSA_DECRYPT Structure" (Part 2: Structures)
TPM_RC
TPMT_RSA_DECRYPT_Unmarshal(
    TPMT_RSA_DECRYPT* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_RSA_DECRYPT_Unmarshal(
        (TPMI_ALG_RSA_DECRYPT*)&(target->scheme), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_ASYM_SCHEME_Unmarshal((TPMU_ASYM_SCHEME*)&(target->details),
                                            buffer,
                                            size,
                                            (UINT32)target->scheme);
    return result;
}
UINT16
TPMT_RSA_DECRYPT_Marshal(TPMT_RSA_DECRYPT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                     + TPMI_ALG_RSA_DECRYPT_Marshal(
                         (TPMI_ALG_RSA_DECRYPT*)&(source->scheme), buffer, size));
    result =
        (UINT16)(result
                 + TPMU_ASYM_SCHEME_Marshal((TPMU_ASYM_SCHEME*)&(source->details),
                                            buffer,
                                            size,
                                            (UINT32)source->scheme));
    return result;
}

// Table "Definition of TPM2B_PUBLIC_KEY_RSA Structure" (Part 2: Structures)
TPM_RC
TPM2B_PUBLIC_KEY_RSA_Unmarshal(
    TPM2B_PUBLIC_KEY_RSA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > MAX_RSA_KEY_BYTES))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
```

```
    return result;
}
UINT16
TPM2B_PUBLIC_KEY_RSA_Marshal(TPM2B_PUBLIC_KEY_RSA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMI_RSA_KEY_BITS Type" (Part 2: Structures)
TPM_RC
TPMI_RSA_KEY_BITS_Unmarshal(TPMI_RSA_KEY_BITS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_KEY_BITS_Unmarshal((TPM_KEY_BITS*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#   if RSA_1024
            case 1024:
#   endif  // RSA_1024
#   if RSA_16384
            case 16384:
#   endif  // RSA_16384
#   if RSA_2048
            case 2048:
#   endif  // RSA_2048
#   if RSA_3072
            case 3072:
#   endif  // RSA_3072
#   if RSA_4096
            case 4096:
#   endif  // RSA_4096
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#   if !USE_MARSHALING_DEFINES
UINT16
TPMI_RSA_KEY_BITS_Marshal(TPMI_RSA_KEY_BITS* source, BYTE** buffer, INT32* size)
{
    return TPM_KEY_BITS_Marshal((TPM_KEY_BITS*)source, buffer, size);
}
#   endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM2B_PRIVATE_KEY_RSA Structure" (Part 2: Structures)
TPM_RC
TPM2B_PRIVATE_KEY_RSA_Unmarshal(
    TPM2B_PRIVATE_KEY_RSA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
```

```
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > RSA_PRIVATE_SIZE))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_PRIVATE_KEY_RSA_Marshal(
    TPM2B_PRIVATE_KEY_RSA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_ECC_PARAMETER Structure" (Part 2: Structures)
#  if ALG_ECC
TPM_RC
TPM2B_ECC_PARAMETER_Unmarshal(TPM2B_ECC_PARAMETER* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > MAX_ECC_KEY_BYTES))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_ECC_PARAMETER_Marshal(TPM2B_ECC_PARAMETER* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of TPMS_ECC_POINT Structure" (Part 2: Structures)
#  if ALG_ECC
TPM_RC
TPMS_ECC_POINT_Unmarshal(TPMS_ECC_POINT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM2B_ECC_PARAMETER_Unmarshal(
        (TPM2B_ECC_PARAMETER*)&(target->x), buffer, size);
```

```
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_ECC_PARAMETER_Unmarshal(
            (TPM2B_ECC_PARAMETER*)&(target->y), buffer, size);
    return result;
}
UINT16
TPMS_ECC_POINT_Marshal(TPMS_ECC_POINT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->x), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->y), buffer, size));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of TPM2B_ECC_POINT Structure" (Part 2: Structures)
#  if ALG_ECC
TPM_RC
TPM2B_ECC_POINT_Unmarshal(TPM2B_ECC_POINT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->size), buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // if size is zero, then the required structure is missing
        if(target->size == 0)
            result = TPM_RC_SIZE;
        else
        {
            INT32 startSize = *size;
            result          = TPMS_ECC_POINT_Unmarshal(
                (TPMS_ECC_POINT*)&(target->point), buffer, size);
            if((result == TPM_RC_SUCCESS) && (target->size != (startSize - *size)))
                result = TPM_RC_SIZE;
        }
    }
    return result;
}
UINT16
TPM2B_ECC_POINT_Marshal(TPM2B_ECC_POINT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    // Marshal a dummy value of the 2B size. This makes sure that 'buffer'
    // and 'size' are advanced as necessary (i.e., if they are present)
    result = UINT16_Marshal(&result, buffer, size);
    // Marshal the structure
    result = (UINT16)(result
                      + TPMS_ECC_POINT_Marshal(
                          (TPMS_ECC_POINT*)&(source->point), buffer, size));
    // if a buffer was provided, go back and fill in the actual size
    if(buffer != NULL)
        UINT16_TO_BYTE_ARRAY((result - 2), (*buffer - result));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of TPMI_ALG_ECC_SCHEME Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_ECC_SCHEME_Unmarshal(
    TPMI_ALG_ECC_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
```

```
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_ECDSA
            case TPM_ALG_ECDSA:
#  endif  // ALG_ECDSA
#  if ALG_ECDH
            case TPM_ALG_ECDH:
#  endif  // ALG_ECDH
#  if ALG_ECDAA
            case TPM_ALG_ECDAA:
#  endif  // ALG_ECDAA
#  if ALG_SM2
            case TPM_ALG_SM2:
#  endif  // ALG_SM2
#  if ALG_ECSCHNORR
            case TPM_ALG_ECSCHNORR:
#  endif  // ALG_ECSCHNORR
#  if ALG_ECMQV
            case TPM_ALG_ECMQV:
#  endif  // ALG_ECMQV
#  if ALG_EDDSA
            case TPM_ALG_EDDSA:
#  endif  // ALG_EDDSA
#  if ALG_EDDSA_PH
            case TPM_ALG_EDDSA_PH:
#  endif  // ALG_EDDSA_PH
                break;
            default:
                if((*target != TPM_ALG_NULL) || !flag)
                    result = TPM_RC_SCHEME;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_ECC_SCHEME_Marshal(TPMI_ALG_ECC_SCHEME* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMI_ECC_CURVE Type" (Part 2: Structures)
TPM_RC
TPMI_ECC_CURVE_Unmarshal(
    TPMI_ECC_CURVE* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPM_ECC_CURVE_Unmarshal((TPM_ECC_CURVE*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ECC_NIST_P192
            case TPM_ECC_NIST_P192:
#  endif  // ECC_NIST_P192
#  if ECC_NIST_P224
            case TPM_ECC_NIST_P224:
#  endif  // ECC_NIST_P224
#  if ECC_NIST_P256
            case TPM_ECC_NIST_P256:
#  endif  // ECC_NIST_P256
```

```
#  if ECC_NIST_P384
            case TPM_ECC_NIST_P384:
#  endif  // ECC_NIST_P384
#  if ECC_NIST_P521
            case TPM_ECC_NIST_P521:
#  endif  // ECC_NIST_P521
#  if ECC_BN_P256
            case TPM_ECC_BN_P256:
#  endif  // ECC_BN_P256
#  if ECC_BN_P638
            case TPM_ECC_BN_P638:
#  endif  // ECC_BN_P638
#  if ECC_SM2_P256
            case TPM_ECC_SM2_P256:
#  endif  // ECC_SM2_P256
#  if ECC_BP_P256_R1
            case TPM_ECC_BP_P256_R1:
#  endif  // ECC_BP_P256_R1
#  if ECC_BP_P384_R1
            case TPM_ECC_BP_P384_R1:
#  endif  // ECC_BP_P384_R1
#  if ECC_BP_P512_R1
            case TPM_ECC_BP_P512_R1:
#  endif  // ECC_BP_P512_R1
#  if ECC_CURVE_25519
            case TPM_ECC_CURVE_25519:
#  endif  // ECC_CURVE_25519
#  if ECC_CURVE_448
            case TPM_ECC_CURVE_448:
#  endif  // ECC_CURVE_448
                break;
            default:
                if((*target != TPM_ECC_NONE) || !flag)
                    result = TPM_RC_CURVE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ECC_CURVE_Marshal(TPMI_ECC_CURVE* source, BYTE** buffer, INT32* size)
{
    return TPM_ECC_CURVE_Marshal((TPM_ECC_CURVE*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMT_ECC_SCHEME Structure" (Part 2: Structures)
#  if ALG_ECC
TPM_RC
TPMT_ECC_SCHEME_Unmarshal(
    TPMT_ECC_SCHEME* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_ECC_SCHEME_Unmarshal(
        (TPMI_ALG_ECC_SCHEME*)&(target->scheme), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_ASYM_SCHEME_Unmarshal((TPMU_ASYM_SCHEME*)&(target->details),
                                            buffer,
                                            size,
                                            (UINT32)target->scheme);
    return result;
}
UINT16
TPMT_ECC_SCHEME_Marshal(TPMT_ECC_SCHEME* source, BYTE** buffer, INT32* size)
{
```

```
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_ECC_SCHEME_Marshal(
                          (TPMI_ALG_ECC_SCHEME*)&(source->scheme), buffer, size));
    result =
        (UINT16)(result
                 + TPMU_ASYM_SCHEME_Marshal((TPMU_ASYM_SCHEME*)&(source->details),
                                            buffer,
                                            size,
                                            (UINT32)source->scheme));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of TPMS_ALGORITHM_DETAIL_ECC Structure" (Part 2: Structures)
#  if ALG_ECC
UINT16
TPMS_ALGORITHM_DETAIL_ECC_Marshal(
    TPMS_ALGORITHM_DETAIL_ECC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPM_ECC_CURVE_Marshal(
                          (TPM_ECC_CURVE*)&(source->curveID), buffer, size));
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->keySize), buffer, size));
    result = (UINT16)(result
                      + TPMT_KDF_SCHEME_Marshal(
                          (TPMT_KDF_SCHEME*)&(source->kdf), buffer, size));
    result = (UINT16)(result
                      + TPMT_ECC_SCHEME_Marshal(
                          (TPMT_ECC_SCHEME*)&(source->sign), buffer, size));
    result = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->p), buffer, size));
    result = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->a), buffer, size));
    result = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->b), buffer, size));
    result = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->gX), buffer, size));
    result = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->gY), buffer, size));
    result = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->n), buffer, size));
    result = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                          (TPM2B_ECC_PARAMETER*)&(source->h), buffer, size));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of TPMS_SIGNATURE_RSA Structure" (Part 2: Structures)
TPM_RC
TPMS_SIGNATURE_RSA_Unmarshal(TPMS_SIGNATURE_RSA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result =
        TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH*)&(target->hash), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_PUBLIC_KEY_RSA_Unmarshal(
```

```
            (TPM2B_PUBLIC_KEY_RSA*)&(target->sig), buffer, size);
    return result;
}
UINT16
TPMS_SIGNATURE_RSA_Marshal(TPMS_SIGNATURE_RSA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
                          (TPMI_ALG_HASH*)&(source->hash), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_PUBLIC_KEY_RSA_Marshal(
                          (TPM2B_PUBLIC_KEY_RSA*)&(source->sig), buffer, size));
    return result;
}

// Table "Definition of Types for Signature" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_RSASSA_Unmarshal(
    TPMS_SIGNATURE_RSASSA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_RSA_Unmarshal((TPMS_SIGNATURE_RSA*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_RSASSA_Marshal(
    TPMS_SIGNATURE_RSASSA* source, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_RSA_Marshal((TPMS_SIGNATURE_RSA*)source, buffer, size);
}
TPM_RC
TPMS_SIGNATURE_RSAPSS_Unmarshal(
    TPMS_SIGNATURE_RSAPSS* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_RSA_Unmarshal((TPMS_SIGNATURE_RSA*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_RSAPSS_Marshal(
    TPMS_SIGNATURE_RSAPSS* source, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_RSA_Marshal((TPMS_SIGNATURE_RSA*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_SIGNATURE_ECC Structure" (Part 2: Structures)
#  if ALG_ECC
TPM_RC
TPMS_SIGNATURE_ECC_Unmarshal(TPMS_SIGNATURE_ECC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result =
        TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH*)&(target->hash), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_ECC_PARAMETER_Unmarshal(
            (TPM2B_ECC_PARAMETER*)&(target->signatureR), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_ECC_PARAMETER_Unmarshal(
            (TPM2B_ECC_PARAMETER*)&(target->signatureS), buffer, size);
    return result;
}
UINT16
TPMS_SIGNATURE_ECC_Marshal(TPMS_SIGNATURE_ECC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
```

```
                        (TPMI_ALG_HASH*)&(source->hash), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                        (TPM2B_ECC_PARAMETER*)&(source->signatureR), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_ECC_PARAMETER_Marshal(
                        (TPM2B_ECC_PARAMETER*)&(source->signatureS), buffer, size));
    return result;
}
#  endif  // ALG_ECC

// Table "Definition of Types for TPMS_SIGNATURE_ECC" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
TPM_RC
TPMS_SIGNATURE_ECDSA_Unmarshal(
    TPMS_SIGNATURE_ECDSA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_ECDSA_Marshal(TPMS_SIGNATURE_ECDSA* source, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)source, buffer, size);
}
TPM_RC
TPMS_SIGNATURE_ECDAA_Unmarshal(
    TPMS_SIGNATURE_ECDAA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_ECDAA_Marshal(TPMS_SIGNATURE_ECDAA* source, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)source, buffer, size);
}
TPM_RC
TPMS_SIGNATURE_SM2_Unmarshal(TPMS_SIGNATURE_SM2* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_SM2_Marshal(TPMS_SIGNATURE_SM2* source, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)source, buffer, size);
}
TPM_RC
TPMS_SIGNATURE_ECSCHNORR_Unmarshal(
    TPMS_SIGNATURE_ECSCHNORR* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_ECSCHNORR_Marshal(
    TPMS_SIGNATURE_ECSCHNORR* source, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)source, buffer, size);
}
TPM_RC
TPMS_SIGNATURE_EDDSA_Unmarshal(
    TPMS_SIGNATURE_EDDSA* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_EDDSA_Marshal(TPMS_SIGNATURE_EDDSA* source, BYTE** buffer, INT32* size)
{
```

```
    return TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)source, buffer, size);
}
TPM_RC
TPMS_SIGNATURE_EDDSA_PH_Unmarshal(
    TPMS_SIGNATURE_EDDSA_PH* target, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Unmarshal((TPMS_SIGNATURE_ECC*)target, buffer, size);
}
UINT16
TPMS_SIGNATURE_EDDSA_PH_Marshal(
    TPMS_SIGNATURE_EDDSA_PH* source, BYTE** buffer, INT32* size)
{
    return TPMS_SIGNATURE_ECC_Marshal((TPMS_SIGNATURE_ECC*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_SIGNATURE Union" (Part 2: Structures)
TPM_RC
TPMU_SIGNATURE_Unmarshal(
    TPMU_SIGNATURE* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_HMAC
        case TPM_ALG_HMAC:
            return TPMT_HA_Unmarshal((TPMT_HA*)&(target->hmac), buffer, size, 0);
#  endif  // ALG_HMAC
#  if ALG_RSASSA
        case TPM_ALG_RSASSA:
            return TPMS_SIGNATURE_RSASSA_Unmarshal(
                (TPMS_SIGNATURE_RSASSA*)&(target->rsassa), buffer, size);
#  endif  // ALG_RSASSA
#  if ALG_RSAPSS
        case TPM_ALG_RSAPSS:
            return TPMS_SIGNATURE_RSAPSS_Unmarshal(
                (TPMS_SIGNATURE_RSAPSS*)&(target->rsapss), buffer, size);
#  endif  // ALG_RSAPSS
#  if ALG_ECDSA
        case TPM_ALG_ECDSA:
            return TPMS_SIGNATURE_ECDSA_Unmarshal(
                (TPMS_SIGNATURE_ECDSA*)&(target->ecdsa), buffer, size);
#  endif  // ALG_ECDSA
#  if ALG_ECDAA
        case TPM_ALG_ECDAA:
            return TPMS_SIGNATURE_ECDAA_Unmarshal(
                (TPMS_SIGNATURE_ECDAA*)&(target->ecdaa), buffer, size);
#  endif  // ALG_ECDAA
#  if ALG_SM2
        case TPM_ALG_SM2:
            return TPMS_SIGNATURE_SM2_Unmarshal(
                (TPMS_SIGNATURE_SM2*)&(target->sm2), buffer, size);
#  endif  // ALG_SM2
#  if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            return TPMS_SIGNATURE_ECSCHNORR_Unmarshal(
                (TPMS_SIGNATURE_ECSCHNORR*)&(target->ecschnorr), buffer, size);
#  endif  // ALG_ECSCHNORR
#  if ALG_EDDSA
        case TPM_ALG_EDDSA:
            return TPMS_SIGNATURE_EDDSA_Unmarshal(
                (TPMS_SIGNATURE_EDDSA*)&(target->eddsa), buffer, size);
#  endif  // ALG_EDDSA
#  if ALG_EDDSA_PH
        case TPM_ALG_EDDSA_PH:
            return TPMS_SIGNATURE_EDDSA_PH_Unmarshal(
                (TPMS_SIGNATURE_EDDSA_PH*)&(target->eddsa_ph), buffer, size);
```

```
#  endif  // ALG_EDDSA_PH
#  if ALG_LMS
        case TPM_ALG_LMS:
            return TPMS_SIGNATURE_LMS_Unmarshal(
                (TPMS_SIGNATURE_LMS*)&(target->lms), buffer, size);
#  endif  // ALG_LMS
#  if ALG_XMSS
        case TPM_ALG_XMSS:
            return TPMS_SIGNATURE_XMSS_Unmarshal(
                (TPMS_SIGNATURE_XMSS*)&(target->xmss), buffer, size);
#  endif  // ALG_XMSS
        case TPM_ALG_NULL:
            return TPM_RC_SUCCESS;
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_SIGNATURE_Marshal(
    TPMU_SIGNATURE* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_HMAC
        case TPM_ALG_HMAC:
            return TPMT_HA_Marshal((TPMT_HA*)&(source->hmac), buffer, size);
#  endif  // ALG_HMAC
#  if ALG_RSASSA
        case TPM_ALG_RSASSA:
            return TPMS_SIGNATURE_RSASSA_Marshal(
                (TPMS_SIGNATURE_RSASSA*)&(source->rsassa), buffer, size);
#  endif  // ALG_RSASSA
#  if ALG_RSAPSS
        case TPM_ALG_RSAPSS:
            return TPMS_SIGNATURE_RSAPSS_Marshal(
                (TPMS_SIGNATURE_RSAPSS*)&(source->rsapss), buffer, size);
#  endif  // ALG_RSAPSS
#  if ALG_ECDSA
        case TPM_ALG_ECDSA:
            return TPMS_SIGNATURE_ECDSA_Marshal(
                (TPMS_SIGNATURE_ECDSA*)&(source->ecdsa), buffer, size);
#  endif  // ALG_ECDSA
#  if ALG_ECDAA
        case TPM_ALG_ECDAA:
            return TPMS_SIGNATURE_ECDAA_Marshal(
                (TPMS_SIGNATURE_ECDAA*)&(source->ecdaa), buffer, size);
#  endif  // ALG_ECDAA
#  if ALG_SM2
        case TPM_ALG_SM2:
            return TPMS_SIGNATURE_SM2_Marshal(
                (TPMS_SIGNATURE_SM2*)&(source->sm2), buffer, size);
#  endif  // ALG_SM2
#  if ALG_ECSCHNORR
        case TPM_ALG_ECSCHNORR:
            return TPMS_SIGNATURE_ECSCHNORR_Marshal(
                (TPMS_SIGNATURE_ECSCHNORR*)&(source->ecschnorr), buffer, size);
#  endif  // ALG_ECSCHNORR
#  if ALG_EDDSA
        case TPM_ALG_EDDSA:
            return TPMS_SIGNATURE_EDDSA_Marshal(
                (TPMS_SIGNATURE_EDDSA*)&(source->eddsa), buffer, size);
#  endif  // ALG_EDDSA
#  if ALG_EDDSA_PH
        case TPM_ALG_EDDSA_PH:
            return TPMS_SIGNATURE_EDDSA_PH_Marshal(
                (TPMS_SIGNATURE_EDDSA_PH*)&(source->eddsa_ph), buffer, size);
#  endif  // ALG_EDDSA_PH
```

```
#  if ALG_LMS
        case TPM_ALG_LMS:
            return TPMS_SIGNATURE_LMS_Marshal(
                (TPMS_SIGNATURE_LMS*)&(source->lms), buffer, size);
#  endif  // ALG_LMS
#  if ALG_XMSS
        case TPM_ALG_XMSS:
            return TPMS_SIGNATURE_XMSS_Marshal(
                (TPMS_SIGNATURE_XMSS*)&(source->xmss), buffer, size);
#  endif  // ALG_XMSS
    }
    return 0;
}

// Table "Definition of TPMT_SIGNATURE Structure" (Part 2: Structures)
TPM_RC
TPMT_SIGNATURE_Unmarshal(
    TPMT_SIGNATURE* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = TPMI_ALG_SIG_SCHEME_Unmarshal(
        (TPMI_ALG_SIG_SCHEME*)&(target->sigAlg), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SIGNATURE_Unmarshal((TPMU_SIGNATURE*)&(target->signature),
                                          buffer,
                                          size,
                                          (UINT32)target->sigAlg);
    return result;
}
UINT16
TPMT_SIGNATURE_Marshal(TPMT_SIGNATURE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_SIG_SCHEME_Marshal(
                          (TPMI_ALG_SIG_SCHEME*)&(source->sigAlg), buffer, size));
    result        = (UINT16)(result
                      + TPMU_SIGNATURE_Marshal((TPMU_SIGNATURE*)&(source->signature),
                                               buffer,
                                               size,
                                               (UINT32)source->sigAlg));
    return result;
}

// Table "Definition of TPMU_ENCRYPTED_SECRET Union" (Part 2: Structures)
TPM_RC
TPMU_ENCRYPTED_SECRET_Unmarshal(
    TPMU_ENCRYPTED_SECRET* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_ECC
        case TPM_ALG_ECC:
            return BYTE_Array_Unmarshal(
                (BYTE*)&(target->ecc), buffer, size, (INT32)sizeof(TPMS_ECC_POINT));
#  endif  // ALG_ECC
#  if ALG_RSA
        case TPM_ALG_RSA:
            return BYTE_Array_Unmarshal(
                (BYTE*)&(target->rsa), buffer, size, (INT32)MAX_RSA_KEY_BYTES);
#  endif  // ALG_RSA
#  if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return BYTE_Array_Unmarshal((BYTE*)&(target->symmetric),
                                        buffer,
                                        size,
```

```
                                        (INT32)sizeof(TPM2B_DIGEST));
#   endif  // ALG_SYMCIPHER
#   if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return BYTE_Array_Unmarshal((BYTE*)&(target->keyedHash),
                                        buffer,
                                        size,
                                        (INT32)sizeof(TPM2B_DIGEST));
#   endif  // ALG_KEYEDHASH
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_ENCRYPTED_SECRET_Marshal(
    TPMU_ENCRYPTED_SECRET* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_ECC
        case TPM_ALG_ECC:
            return BYTE_Array_Marshal(
                (BYTE*)&(source->ecc), buffer, size, (INT32)sizeof(TPMS_ECC_POINT));
#   endif  // ALG_ECC
#   if ALG_RSA
        case TPM_ALG_RSA:
            return BYTE_Array_Marshal(
                (BYTE*)&(source->rsa), buffer, size, (INT32)MAX_RSA_KEY_BYTES);
#   endif  // ALG_RSA
#   if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return BYTE_Array_Marshal((BYTE*)&(source->symmetric),
                                      buffer,
                                      size,
                                      (INT32)sizeof(TPM2B_DIGEST));
#   endif  // ALG_SYMCIPHER
#   if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return BYTE_Array_Marshal((BYTE*)&(source->keyedHash),
                                      buffer,
                                      size,
                                      (INT32)sizeof(TPM2B_DIGEST));
#   endif  // ALG_KEYEDHASH
    }
    return 0;
}

// Table "Definition of TPM2B_ENCRYPTED_SECRET Structure" (Part 2: Structures)
TPM_RC
TPM2B_ENCRYPTED_SECRET_Unmarshal(
    TPM2B_ENCRYPTED_SECRET* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMU_ENCRYPTED_SECRET)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.secret), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_ENCRYPTED_SECRET_Marshal(
    TPM2B_ENCRYPTED_SECRET* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
```

```
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.secret),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMI_ALG_PUBLIC Type" (Part 2: Structures)
TPM_RC
TPMI_ALG_PUBLIC_Unmarshal(TPMI_ALG_PUBLIC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_ALG_ID_Unmarshal((TPM_ALG_ID*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
#  if ALG_RSA
            case TPM_ALG_RSA:
#  endif  // ALG_RSA
#  if ALG_KEYEDHASH
            case TPM_ALG_KEYEDHASH:
#  endif  // ALG_KEYEDHASH
#  if ALG_ECC
            case TPM_ALG_ECC:
#  endif  // ALG_ECC
#  if ALG_SYMCIPHER
            case TPM_ALG_SYMCIPHER:
#  endif  // ALG_SYMCIPHER
                break;
            default:
                result = TPM_RC_TYPE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMI_ALG_PUBLIC_Marshal(TPMI_ALG_PUBLIC* source, BYTE** buffer, INT32* size)
{
    return TPM_ALG_ID_Marshal((TPM_ALG_ID*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMU_PUBLIC_ID Union" (Part 2: Structures)
TPM_RC
TPMU_PUBLIC_ID_Unmarshal(
    TPMU_PUBLIC_ID* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPM2B_DIGEST_Unmarshal(
                (TPM2B_DIGEST*)&(target->keyedHash), buffer, size);
#  endif  // ALG_KEYEDHASH
#  if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPM2B_DIGEST_Unmarshal(
                (TPM2B_DIGEST*)&(target->sym), buffer, size);
```

```
#  endif  // ALG_SYMCIPHER
#  if ALG_RSA
        case TPM_ALG_RSA:
            return TPM2B_PUBLIC_KEY_RSA_Unmarshal(
                (TPM2B_PUBLIC_KEY_RSA*)&(target->rsa), buffer, size);
#  endif  // ALG_RSA
#  if ALG_ECC
        case TPM_ALG_ECC:
            return TPMS_ECC_POINT_Unmarshal(
                (TPMS_ECC_POINT*)&(target->ecc), buffer, size);
#  endif  // ALG_ECC
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_PUBLIC_ID_Marshal(
    TPMU_PUBLIC_ID* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPM2B_DIGEST_Marshal(
                (TPM2B_DIGEST*)&(source->keyedHash), buffer, size);
#  endif  // ALG_KEYEDHASH
#  if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPM2B_DIGEST_Marshal((TPM2B_DIGEST*)&(source->sym), buffer, size);
#  endif  // ALG_SYMCIPHER
#  if ALG_RSA
        case TPM_ALG_RSA:
            return TPM2B_PUBLIC_KEY_RSA_Marshal(
                (TPM2B_PUBLIC_KEY_RSA*)&(source->rsa), buffer, size);
#  endif  // ALG_RSA
#  if ALG_ECC
        case TPM_ALG_ECC:
            return TPMS_ECC_POINT_Marshal(
                (TPMS_ECC_POINT*)&(source->ecc), buffer, size);
#  endif  // ALG_ECC
    }
    return 0;
}

// Table "Definition of TPMS_KEYEDHASH_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_KEYEDHASH_PARMS_Unmarshal(
    TPMS_KEYEDHASH_PARMS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPMT_KEYEDHASH_SCHEME_Unmarshal(
        (TPMT_KEYEDHASH_SCHEME*)&(target->scheme), buffer, size, 1);
    return result;
}
UINT16
TPMS_KEYEDHASH_PARMS_Marshal(TPMS_KEYEDHASH_PARMS* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                    + TPMT_KEYEDHASH_SCHEME_Marshal(
                        (TPMT_KEYEDHASH_SCHEME*)&(source->scheme), buffer, size));
    return result;
}

// Table "Definition of TPMS_RSA_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_RSA_PARMS_Unmarshal(TPMS_RSA_PARMS* target, BYTE** buffer, INT32* size)
```

```
{
    TPM_RC result;
    result = TPMT_SYM_DEF_OBJECT_Unmarshal(
        (TPMT_SYM_DEF_OBJECT*)&(target->symmetric), buffer, size, 1);
    if(result == TPM_RC_SUCCESS)
        result = TPMT_RSA_SCHEME_Unmarshal(
            (TPMT_RSA_SCHEME*)&(target->scheme), buffer, size, 1);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_RSA_KEY_BITS_Unmarshal(
            (TPMI_RSA_KEY_BITS*)&(target->keyBits), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = UINT32_Unmarshal((UINT32*)&(target->exponent), buffer, size);
    return result;
}
UINT16
TPMS_RSA_PARMS_Marshal(TPMS_RSA_PARMS* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMT_SYM_DEF_OBJECT_Marshal(
                          (TPMT_SYM_DEF_OBJECT*)&(source->symmetric), buffer, size));
    result        = (UINT16)(result
                      + TPMT_RSA_SCHEME_Marshal(
                          (TPMT_RSA_SCHEME*)&(source->scheme), buffer, size));
    result        = (UINT16)(result
                      + TPMI_RSA_KEY_BITS_Marshal(
                          (TPMI_RSA_KEY_BITS*)&(source->keyBits), buffer, size));
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->exponent), buffer, size));
    return result;
}

// Table "Definition of TPMS_ECC_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMS_ECC_PARMS_Unmarshal(TPMS_ECC_PARMS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPMT_SYM_DEF_OBJECT_Unmarshal(
        (TPMT_SYM_DEF_OBJECT*)&(target->symmetric), buffer, size, 1);
    if(result == TPM_RC_SUCCESS)
        result = TPMT_ECC_SCHEME_Unmarshal(
            (TPMT_ECC_SCHEME*)&(target->scheme), buffer, size, 1);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_ECC_CURVE_Unmarshal(
            (TPMI_ECC_CURVE*)&(target->curveID), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
        result = TPMT_KDF_SCHEME_Unmarshal(
            (TPMT_KDF_SCHEME*)&(target->kdf), buffer, size, 1);
    return result;
}
UINT16
TPMS_ECC_PARMS_Marshal(TPMS_ECC_PARMS* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMT_SYM_DEF_OBJECT_Marshal(
                          (TPMT_SYM_DEF_OBJECT*)&(source->symmetric), buffer, size));
    result        = (UINT16)(result
                      + TPMT_ECC_SCHEME_Marshal(
                          (TPMT_ECC_SCHEME*)&(source->scheme), buffer, size));
    result        = (UINT16)(result
                      + TPMI_ECC_CURVE_Marshal(
                          (TPMI_ECC_CURVE*)&(source->curveID), buffer, size));
    result        = (UINT16)(result
                      + TPMT_KDF_SCHEME_Marshal(
                          (TPMT_KDF_SCHEME*)&(source->kdf), buffer, size));
```

```
    return result;
}

// Table "Definition of TPMU_PUBLIC_PARMS Union" (Part 2: Structures)
TPM_RC
TPMU_PUBLIC_PARMS_Unmarshal(
    TPMU_PUBLIC_PARMS* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPMS_KEYEDHASH_PARMS_Unmarshal(
                (TPMS_KEYEDHASH_PARMS*)&(target->keyedHashDetail), buffer, size);
#   endif  // ALG_KEYEDHASH
#   if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPMS_SYMCIPHER_PARMS_Unmarshal(
                (TPMS_SYMCIPHER_PARMS*)&(target->symDetail), buffer, size);
#   endif  // ALG_SYMCIPHER
#   if ALG_RSA
        case TPM_ALG_RSA:
            return TPMS_RSA_PARMS_Unmarshal(
                (TPMS_RSA_PARMS*)&(target->rsaDetail), buffer, size);
#   endif  // ALG_RSA
#   if ALG_ECC
        case TPM_ALG_ECC:
            return TPMS_ECC_PARMS_Unmarshal(
                (TPMS_ECC_PARMS*)&(target->eccDetail), buffer, size);
#   endif  // ALG_ECC
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_PUBLIC_PARMS_Marshal(
    TPMU_PUBLIC_PARMS* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPMS_KEYEDHASH_PARMS_Marshal(
                (TPMS_KEYEDHASH_PARMS*)&(source->keyedHashDetail), buffer, size);
#   endif  // ALG_KEYEDHASH
#   if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPMS_SYMCIPHER_PARMS_Marshal(
                (TPMS_SYMCIPHER_PARMS*)&(source->symDetail), buffer, size);
#   endif  // ALG_SYMCIPHER
#   if ALG_RSA
        case TPM_ALG_RSA:
            return TPMS_RSA_PARMS_Marshal(
                (TPMS_RSA_PARMS*)&(source->rsaDetail), buffer, size);
#   endif  // ALG_RSA
#   if ALG_ECC
        case TPM_ALG_ECC:
            return TPMS_ECC_PARMS_Marshal(
                (TPMS_ECC_PARMS*)&(source->eccDetail), buffer, size);
#   endif  // ALG_ECC
    }
    return 0;
}

// Table "Definition of TPMT_PUBLIC_PARMS Structure" (Part 2: Structures)
TPM_RC
TPMT_PUBLIC_PARMS_Unmarshal(TPMT_PUBLIC_PARMS* target, BYTE** buffer, INT32* size)
```

```
{
    TPM_RC result;
    result =
        TPMI_ALG_PUBLIC_Unmarshal((TPMI_ALG_PUBLIC*)&(target->type), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPMU_PUBLIC_PARMS_Unmarshal((TPMU_PUBLIC_PARMS*)&(target->parameters),
                                        buffer,
                                        size,
                                        (UINT32)target->type);
    return result;
}
UINT16
TPMT_PUBLIC_PARMS_Marshal(TPMT_PUBLIC_PARMS* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_PUBLIC_Marshal(
                          (TPMI_ALG_PUBLIC*)&(source->type), buffer, size));
    result        = (UINT16)(result
                      + TPMU_PUBLIC_PARMS_Marshal(
                          (TPMU_PUBLIC_PARMS*)&(source->parameters),
                          buffer,
                          size,
                          (UINT32)source->type));
    return result;
}

// Table "Definition of TPMT_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPMT_PUBLIC_Unmarshal(TPMT_PUBLIC* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result =
        TPMI_ALG_PUBLIC_Unmarshal((TPMI_ALG_PUBLIC*)&(target->type), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_ALG_HASH_Unmarshal(
            (TPMI_ALG_HASH*)&(target->nameAlg), buffer, size, flag);
    if(result == TPM_RC_SUCCESS)
        result = TPMA_OBJECT_Unmarshal(
            (TPMA_OBJECT*)&(target->objectAttributes), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_DIGEST_Unmarshal(
            (TPM2B_DIGEST*)&(target->authPolicy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPMU_PUBLIC_PARMS_Unmarshal((TPMU_PUBLIC_PARMS*)&(target->parameters),
                                        buffer,
                                        size,
                                        (UINT32)target->type);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_PUBLIC_ID_Unmarshal(
            (TPMU_PUBLIC_ID*)&(target->unique), buffer, size, (UINT32)target->type);
    return result;
}
UINT16
TPMT_PUBLIC_Marshal(TPMT_PUBLIC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_PUBLIC_Marshal(
                          (TPMI_ALG_PUBLIC*)&(source->type), buffer, size));
    result        = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
                          (TPMI_ALG_HASH*)&(source->nameAlg), buffer, size));
    result        = (UINT16)(result
```

```
                    + TPMA_OBJECT_Marshal(
                        (TPMA_OBJECT*)&(source->objectAttributes), buffer, size));
    result        = (UINT16)(result
                    + TPM2B_DIGEST_Marshal(
                        (TPM2B_DIGEST*)&(source->authPolicy), buffer, size));
    result        = (UINT16)(result
                    + TPMU_PUBLIC_PARMS_Marshal(
                        (TPMU_PUBLIC_PARMS*)&(source->parameters),
                       buffer,
                        size,
                        (UINT32)source->type));
    result        = (UINT16)(result
                    + TPMU_PUBLIC_ID_Marshal((TPMU_PUBLIC_ID*)&(source->unique),
                                             buffer,
                                             size,
                                             (UINT32)source->type));
    return result;
}

// Table "Definition of TPM2B_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPM2B_PUBLIC_Unmarshal(TPM2B_PUBLIC* target, BYTE** buffer, INT32* size, BOOL flag)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->size), buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // if size is zero, then the required structure is missing
        if(target->size == 0)
            result = TPM_RC_SIZE;
        else
        {
            INT32 startSize = *size;
            result          = TPMT_PUBLIC_Unmarshal(
                (TPMT_PUBLIC*)&(target->publicArea), buffer, size, flag);
            if((result == TPM_RC_SUCCESS) && (target->size != (startSize - *size)))
                result = TPM_RC_SIZE;
        }
    }
    return result;
}
UINT16
TPM2B_PUBLIC_Marshal(TPM2B_PUBLIC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    // Marshal a dummy value of the 2B size. This makes sure that 'buffer'
    // and 'size' are advanced as necessary (i.e., if they are present)
    result = UINT16_Marshal(&result, buffer, size);
    // Marshal the structure
    result = (UINT16)(result
                    + TPMT_PUBLIC_Marshal(
                        (TPMT_PUBLIC*)&(source->publicArea), buffer, size));
    // if a buffer was provided, go back and fill in the actual size
    if(buffer != NULL)
        UINT16_TO_BYTE_ARRAY((result - 2), (*buffer - result));
    return result;
}

// Table "Definition of TPM2B_TEMPLATE Structure" (Part 2: Structures)
TPM_RC
TPM2B_TEMPLATE_Unmarshal(TPM2B_TEMPLATE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMT_PUBLIC)))
        result = TPM_RC_SIZE;
```

```
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_TEMPLATE_Marshal(TPM2B_TEMPLATE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_PRIVATE_VENDOR_SPECIFIC Structure" (Part 2: Structures)
TPM_RC
TPM2B_PRIVATE_VENDOR_SPECIFIC_Unmarshal(
    TPM2B_PRIVATE_VENDOR_SPECIFIC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > PRIVATE_VENDOR_SPECIFIC_BYTES))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_PRIVATE_VENDOR_SPECIFIC_Marshal(
    TPM2B_PRIVATE_VENDOR_SPECIFIC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMU_SENSITIVE_COMPOSITE Union" (Part 2: Structures)
TPM_RC
TPMU_SENSITIVE_COMPOSITE_Unmarshal(
    TPMU_SENSITIVE_COMPOSITE* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#  if ALG_RSA
        case TPM_ALG_RSA:
            return TPM2B_PRIVATE_KEY_RSA_Unmarshal(
                (TPM2B_PRIVATE_KEY_RSA*)&(target->rsa), buffer, size);
#  endif  // ALG_RSA
```

```
#   if ALG_ECC
        case TPM_ALG_ECC:
            return TPM2B_ECC_PARAMETER_Unmarshal(
                (TPM2B_ECC_PARAMETER*)&(target->ecc), buffer, size);
#   endif  // ALG_ECC
#   if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPM2B_SENSITIVE_DATA_Unmarshal(
                (TPM2B_SENSITIVE_DATA*)&(target->bits), buffer, size);
#   endif  // ALG_KEYEDHASH
#   if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPM2B_SYM_KEY_Unmarshal(
                (TPM2B_SYM_KEY*)&(target->sym), buffer, size);
#   endif  // ALG_SYMCIPHER
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_SENSITIVE_COMPOSITE_Marshal(
    TPMU_SENSITIVE_COMPOSITE* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
#   if ALG_RSA
        case TPM_ALG_RSA:
            return TPM2B_PRIVATE_KEY_RSA_Marshal(
                (TPM2B_PRIVATE_KEY_RSA*)&(source->rsa), buffer, size);
#   endif  // ALG_RSA
#   if ALG_ECC
        case TPM_ALG_ECC:
            return TPM2B_ECC_PARAMETER_Marshal(
                (TPM2B_ECC_PARAMETER*)&(source->ecc), buffer, size);
#   endif  // ALG_ECC
#   if ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPM2B_SENSITIVE_DATA_Marshal(
                (TPM2B_SENSITIVE_DATA*)&(source->bits), buffer, size);
#   endif  // ALG_KEYEDHASH
#   if ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPM2B_SYM_KEY_Marshal(
                (TPM2B_SYM_KEY*)&(source->sym), buffer, size);
#   endif  // ALG_SYMCIPHER
    }
    return 0;
}

// Table "Definition of TPMT_SENSITIVE Structure" (Part 2: Structures)
TPM_RC
TPMT_SENSITIVE_Unmarshal(TPMT_SENSITIVE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPMI_ALG_PUBLIC_Unmarshal(
        (TPMI_ALG_PUBLIC*)&(target->sensitiveType), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPM2B_AUTH_Unmarshal((TPM2B_AUTH*)&(target->authValue), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result =
            TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)&(target->seedValue), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_SENSITIVE_COMPOSITE_Unmarshal(
            (TPMU_SENSITIVE_COMPOSITE*)&(target->sensitive),
            buffer,
            size,
```

```
            (UINT32)target->sensitiveType);
    return result;
}
UINT16
TPMT_SENSITIVE_Marshal(TPMT_SENSITIVE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_ALG_PUBLIC_Marshal(
                          (TPMI_ALG_PUBLIC*)&(source->sensitiveType), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_AUTH_Marshal(
                          (TPM2B_AUTH*)&(source->authValue), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->seedValue), buffer, size));
    result        = (UINT16)(result
                      + TPMU_SENSITIVE_COMPOSITE_Marshal(
                          (TPMU_SENSITIVE_COMPOSITE*)&(source->sensitive),
                          buffer,
                          size,
                          (UINT32)source->sensitiveType));
    return result;
}

// Table "Definition of TPM2B_SENSITIVE Structure" (Part 2: Structures)
TPM_RC
TPM2B_SENSITIVE_Unmarshal(TPM2B_SENSITIVE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->size), buffer, size);
    // if there was an error or if target->size equal to 0,
    // skip unmarshaling of the structure
    if((result == TPM_RC_SUCCESS) && (target->size != 0))
    {
        INT32 startSize = *size;
        result          = TPMT_SENSITIVE_Unmarshal(
            (TPMT_SENSITIVE*)&(target->sensitiveArea), buffer, size);
        if((result == TPM_RC_SUCCESS) && (target->size != (startSize - *size)))
            result = TPM_RC_SIZE;
    }
    return result;
}
UINT16
TPM2B_SENSITIVE_Marshal(TPM2B_SENSITIVE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    // Marshal a dummy value of the 2B size. This makes sure that 'buffer'
    // and 'size' are advanced as necessary (i.e., if they are present)
    result = UINT16_Marshal(&result, buffer, size);
    // Marshal the structure
    result = (UINT16)(result
                      + TPMT_SENSITIVE_Marshal(
                          (TPMT_SENSITIVE*)&(source->sensitiveArea), buffer, size));
    // if a buffer was provided, go back and fill in the actual size
    if(buffer != NULL)
        UINT16_TO_BYTE_ARRAY((result - 2), (*buffer - result));
    return result;
}

// Table "Definition of TPM2B_PRIVATE Structure" (Part 2: Structures)
TPM_RC
TPM2B_PRIVATE_Unmarshal(TPM2B_PRIVATE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
```

```
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(_PRIVATE)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_PRIVATE_Marshal(TPM2B_PRIVATE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPM2B_ID_OBJECT Structure" (Part 2: Structures)
TPM_RC
TPM2B_ID_OBJECT_Unmarshal(TPM2B_ID_OBJECT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMS_ID_OBJECT)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.credential), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_ID_OBJECT_Marshal(TPM2B_ID_OBJECT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.credential),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_NV_PIN_COUNTER_PARAMETERS Structure" (Part 2: Structures)
TPM_RC
TPMS_NV_PIN_COUNTER_PARAMETERS_Unmarshal(
    TPMS_NV_PIN_COUNTER_PARAMETERS* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)&(target->pinCount), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = UINT32_Unmarshal((UINT32*)&(target->pinLimit), buffer, size);
    return result;
}
UINT16
```

```
TPMS_NV_PIN_COUNTER_PARAMETERS_Marshal(
    TPMS_NV_PIN_COUNTER_PARAMETERS* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->pinCount), buffer, size));
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->pinLimit), buffer, size));
    return result;
}

// Table "Definition of TPMA_NV Bits" (Part 2: Structures)
TPM_RC
TPMA_NV_Unmarshal(TPMA_NV* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // check that no reserved bits are set
        if(*((UINT32*)target) & (UINT32)0x01f00300)
            result = TPM_RC_RESERVED_BITS;
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_NV_Marshal(TPMA_NV* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMA_NV_EXP Bits" (Part 2: Structures)
TPM_RC
TPMA_NV_EXP_Unmarshal(TPMA_NV_EXP* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT64_Unmarshal((UINT64*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // check that no reserved bits are set
        if(*((UINT64*)target) & (UINT64)0xffffff801f00300)
            result = TPM_RC_RESERVED_BITS;
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPMA_NV_EXP_Marshal(TPMA_NV_EXP* source, BYTE** buffer, INT32* size)
{
    return UINT64_Marshal((UINT64*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_NV_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPMS_NV_PUBLIC_Unmarshal(TPMS_NV_PUBLIC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPMI_RH_NV_LEGACY_INDEX_Unmarshal(
        (TPMI_RH_NV_LEGACY_INDEX*)&(target->nvIndex), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_ALG_HASH_Unmarshal(
            (TPMI_ALG_HASH*)&(target->nameAlg), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
```

```
        result = TPMA_NV_Unmarshal((TPMA_NV*)&(target->attributes), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_DIGEST_Unmarshal(
            (TPM2B_DIGEST*)&(target->authPolicy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = UINT16_Unmarshal((UINT16*)&(target->dataSize), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->dataSize > MAX_NV_INDEX_SIZE))
        result = TPM_RC_SIZE;
    return result;
}
UINT16
TPMS_NV_PUBLIC_Marshal(TPMS_NV_PUBLIC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result
                 + TPMI_RH_NV_LEGACY_INDEX_Marshal(
                     (TPMI_RH_NV_LEGACY_INDEX*)&(source->nvIndex), buffer, size));
    result = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
                          (TPMI_ALG_HASH*)&(source->nameAlg), buffer, size));
    result =
        (UINT16)(result
                 + TPMA_NV_Marshal((TPMA_NV*)&(source->attributes), buffer, size));
    result = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->authPolicy), buffer, size));
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->dataSize), buffer, size));
    return result;
}

// Table "Definition of TPM2B_NV_PUBLIC Structure" (Part 2: Structures)
TPM_RC
TPM2B_NV_PUBLIC_Unmarshal(TPM2B_NV_PUBLIC* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->size), buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // if size is zero, then the required structure is missing
        if(target->size == 0)
            result = TPM_RC_SIZE;
        else
        {
            INT32 startSize = *size;
            result          = TPMS_NV_PUBLIC_Unmarshal(
                (TPMS_NV_PUBLIC*)&(target->nvPublic), buffer, size);
            if((result == TPM_RC_SUCCESS) && (target->size != (startSize - *size)))
                result = TPM_RC_SIZE;
        }
    }
    return result;
}
UINT16
TPM2B_NV_PUBLIC_Marshal(TPM2B_NV_PUBLIC* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    // Marshal a dummy value of the 2B size. This makes sure that 'buffer'
    // and 'size' are advanced as necessary (i.e., if they are present)
    result = UINT16_Marshal(&result, buffer, size);
    // Marshal the structure
    result = (UINT16)(result
                      + TPMS_NV_PUBLIC_Marshal(
                          (TPMS_NV_PUBLIC*)&(source->nvPublic), buffer, size));
    // if a buffer was provided, go back and fill in the actual size
```

```
    if(buffer != NULL)
        UINT16_TO_BYTE_ARRAY((result - 2), (*buffer - result));
    return result;
}

// Table "Definition of TPMS_NV_PUBLIC_EXP_ATTR Structure" (Part 2: Structures)
TPM_RC
TPMS_NV_PUBLIC_EXP_ATTR_Unmarshal(
    TPMS_NV_PUBLIC_EXP_ATTR* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPMI_RH_NV_EXP_INDEX_Unmarshal(
        (TPMI_RH_NV_EXP_INDEX*)&(target->nvIndex), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_ALG_HASH_Unmarshal(
            (TPMI_ALG_HASH*)&(target->nameAlg), buffer, size, 0);
    if(result == TPM_RC_SUCCESS)
        result =
            TPMA_NV_EXP_Unmarshal((TPMA_NV_EXP*)&(target->attributes), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_DIGEST_Unmarshal(
            (TPM2B_DIGEST*)&(target->authPolicy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = UINT16_Unmarshal((UINT16*)&(target->dataSize), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->dataSize > MAX_NV_INDEX_SIZE))
        result = TPM_RC_SIZE;
    return result;
}
UINT16
TPMS_NV_PUBLIC_EXP_ATTR_Marshal(
    TPMS_NV_PUBLIC_EXP_ATTR* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPMI_RH_NV_EXP_INDEX_Marshal(
                          (TPMI_RH_NV_EXP_INDEX*)&(source->nvIndex), buffer, size));
    result        = (UINT16)(result
                      + TPMI_ALG_HASH_Marshal(
                          (TPMI_ALG_HASH*)&(source->nameAlg), buffer, size));
    result        = (UINT16)(result
                      + TPMA_NV_EXP_Marshal(
                          (TPMA_NV_EXP*)&(source->attributes), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->authPolicy), buffer, size));
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->dataSize), buffer, size));
    return result;
}

// Table "Definition of TPMU_NV_PUBLIC_2 Union" (Part 2: Structures)
TPM_RC
TPMU_NV_PUBLIC_2_Unmarshal(
    TPMU_NV_PUBLIC_2* target, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
        case TPM_HT_NV_INDEX:
            return TPMS_NV_PUBLIC_Unmarshal(
                (TPMS_NV_PUBLIC*)&(target->nvIndex), buffer, size);
        case TPM_HT_EXTERNAL_NV:
            return TPMS_NV_PUBLIC_EXP_ATTR_Unmarshal(
                (TPMS_NV_PUBLIC_EXP_ATTR*)&(target->externalNV), buffer, size);
        case TPM_HT_PERMANENT_NV:
            return TPMS_NV_PUBLIC_Unmarshal(
                (TPMS_NV_PUBLIC*)&(target->permanentNV), buffer, size);
```

```
    }
    return TPM_RC_SELECTOR;
}
UINT16
TPMU_NV_PUBLIC_2_Marshal(
    TPMU_NV_PUBLIC_2* source, BYTE** buffer, INT32* size, UINT32 selector)
{
    switch(selector)
    {
        case TPM_HT_NV_INDEX:
            return TPMS_NV_PUBLIC_Marshal(
                (TPMS_NV_PUBLIC*)&(source->nvIndex), buffer, size);
        case TPM_HT_EXTERNAL_NV:
            return TPMS_NV_PUBLIC_EXP_ATTR_Marshal(
                (TPMS_NV_PUBLIC_EXP_ATTR*)&(source->externalNV), buffer, size);
        case TPM_HT_PERMANENT_NV:
            return TPMS_NV_PUBLIC_Marshal(
                (TPMS_NV_PUBLIC*)&(source->permanentNV), buffer, size);
    }
    return 0;
}

// Table "Definition of TPMT_NV_PUBLIC_2 Structure" (Part 2: Structures)
TPM_RC
TPMT_NV_PUBLIC_2_Unmarshal(TPMT_NV_PUBLIC_2* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = TPM_HT_Unmarshal((TPM_HT*)&(target->handleType), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMU_NV_PUBLIC_2_Unmarshal((TPMU_NV_PUBLIC_2*)&(target->nvPublic2),
                                            buffer,
                                            size,
                                            (UINT32)target->handleType);
    return result;
}
UINT16
TPMT_NV_PUBLIC_2_Marshal(TPMT_NV_PUBLIC_2* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPM_HT_Marshal((TPM_HT*)&(source->handleType), buffer, size));
    result =
        (UINT16)(result
                 + TPMU_NV_PUBLIC_2_Marshal((TPMU_NV_PUBLIC_2*)&(source->nvPublic2),
                                            buffer,
                                            size,
                                            (UINT32)source->handleType));
    return result;
}

// Table "Definition of TPM2B_NV_PUBLIC_2 Structure" (Part 2: Structures)
TPM_RC
TPM2B_NV_PUBLIC_2_Unmarshal(TPM2B_NV_PUBLIC_2* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->size), buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        // if size is zero, then the required structure is missing
        if(target->size == 0)
            result = TPM_RC_SIZE;
        else
        {
            INT32 startSize = *size;
            result          = TPMT_NV_PUBLIC_2_Unmarshal(
                (TPMT_NV_PUBLIC_2*)&(target->nvPublic2), buffer, size);
```

```
            if((result == TPM_RC_SUCCESS) && (target->size != (startSize - *size)))
                result = TPM_RC_SIZE;
        }
    }
    return result;
}
UINT16
TPM2B_NV_PUBLIC_2_Marshal(TPM2B_NV_PUBLIC_2* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    // Marshal a dummy value of the 2B size. This makes sure that 'buffer'
    // and 'size' are advanced as necessary (i.e., if they are present)
    result = UINT16_Marshal(&result, buffer, size);
    // Marshal the structure
    result = (UINT16)(result
                      + TPMT_NV_PUBLIC_2_Marshal(
                          (TPMT_NV_PUBLIC_2*)&(source->nvPublic2), buffer, size));
    // if a buffer was provided, go back and fill in the actual size
    if(buffer != NULL)
        UINT16_TO_BYTE_ARRAY((result - 2), (*buffer - result));
    return result;
}

// Table "Definition of TPM2B_CONTEXT_SENSITIVE Structure" (Part 2: Structures)
TPM_RC
TPM2B_CONTEXT_SENSITIVE_Unmarshal(
    TPM2B_CONTEXT_SENSITIVE* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > MAX_CONTEXT_SIZE))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_CONTEXT_SENSITIVE_Marshal(
    TPM2B_CONTEXT_SENSITIVE* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_CONTEXT_DATA Structure" (Part 2: Structures)
TPM_RC
TPMS_CONTEXT_DATA_Unmarshal(TPMS_CONTEXT_DATA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result =
        TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST*)&(target->integrity), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_CONTEXT_SENSITIVE_Unmarshal(
            (TPM2B_CONTEXT_SENSITIVE*)&(target->encrypted), buffer, size);
    return result;
```

```
}
UINT16
TPMS_CONTEXT_DATA_Marshal(TPMS_CONTEXT_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->integrity), buffer, size));
    result =
        (UINT16)(result
                 + TPM2B_CONTEXT_SENSITIVE_Marshal(
                     (TPM2B_CONTEXT_SENSITIVE*)&(source->encrypted), buffer, size));
    return result;
}

// Table "Definition of TPM2B_CONTEXT_DATA Structure" (Part 2: Structures)
TPM_RC
TPM2B_CONTEXT_DATA_Unmarshal(TPM2B_CONTEXT_DATA* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT16_Unmarshal((UINT16*)&(target->t.size), buffer, size);
    if((result == TPM_RC_SUCCESS) && (target->t.size > sizeof(TPMS_CONTEXT_DATA)))
        result = TPM_RC_SIZE;
    if(result == TPM_RC_SUCCESS)
        result = BYTE_Array_Unmarshal(
            (BYTE*)&(target->t.buffer), buffer, size, (INT32)target->t.size);
    return result;
}
UINT16
TPM2B_CONTEXT_DATA_Marshal(TPM2B_CONTEXT_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT16_Marshal((UINT16*)&(source->t.size), buffer, size));
    // if size equal to 0, the rest of the structure is a zero buffer
    if(source->t.size == 0)
        return result;
    result = (UINT16)(result
                      + BYTE_Array_Marshal((BYTE*)&(source->t.buffer),
                                           buffer,
                                           size,
                                           (INT32)source->t.size));
    return result;
}

// Table "Definition of TPMS_CONTEXT Structure" (Part 2: Structures)
TPM_RC
TPMS_CONTEXT_Unmarshal(TPMS_CONTEXT* target, BYTE** buffer, INT32* size)
{
    TPM_RC result;
    result = UINT64_Unmarshal((UINT64*)&(target->sequence), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_DH_SAVED_Unmarshal(
            (TPMI_DH_SAVED*)&(target->savedHandle), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPMI_RH_HIERARCHY_Unmarshal(
            (TPMI_RH_HIERARCHY*)&(target->hierarchy), buffer, size);
    if(result == TPM_RC_SUCCESS)
        result = TPM2B_CONTEXT_DATA_Unmarshal(
            (TPM2B_CONTEXT_DATA*)&(target->contextBlob), buffer, size);
    return result;
}
UINT16
TPMS_CONTEXT_Marshal(TPMS_CONTEXT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
```

```
    result =
        (UINT16)(result + UINT64_Marshal((UINT64*)&(source->sequence), buffer, size));
    result = (UINT16)(result
                      + TPMI_DH_SAVED_Marshal(
                          (TPMI_DH_SAVED*)&(source->savedHandle), buffer, size));
    result = (UINT16)(result
                      + TPMI_RH_HIERARCHY_Marshal(
                          (TPMI_RH_HIERARCHY*)&(source->hierarchy), buffer, size));
    result = (UINT16)(result
                      + TPM2B_CONTEXT_DATA_Marshal(
                          (TPM2B_CONTEXT_DATA*)&(source->contextBlob), buffer, size));
    return result;
}

// Table "Definition of TPMS_CREATION_DATA Structure" (Part 2: Structures)
UINT16
TPMS_CREATION_DATA_Marshal(TPMS_CREATION_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result        = (UINT16)(result
                      + TPML_PCR_SELECTION_Marshal(
                          (TPML_PCR_SELECTION*)&(source->pcrSelect), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_DIGEST_Marshal(
                          (TPM2B_DIGEST*)&(source->pcrDigest), buffer, size));
    result        = (UINT16)(result
                      + TPMA_LOCALITY_Marshal(
                          (TPMA_LOCALITY*)&(source->locality), buffer, size));
    result        = (UINT16)(result
                      + TPM_ALG_ID_Marshal(
                          (TPM_ALG_ID*)&(source->parentNameAlg), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_NAME_Marshal(
                          (TPM2B_NAME*)&(source->parentName), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_NAME_Marshal(
                          (TPM2B_NAME*)&(source->parentQualifiedName), buffer, size));
    result        = (UINT16)(result
                      + TPM2B_DATA_Marshal(
                          (TPM2B_DATA*)&(source->outsideInfo), buffer, size));
    return result;
}

// Table "Definition of TPM2B_CREATION_DATA Structure" (Part 2: Structures)
UINT16
TPM2B_CREATION_DATA_Marshal(TPM2B_CREATION_DATA* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    // Marshal a dummy value of the 2B size. This makes sure that 'buffer'
    // and 'size' are advanced as necessary (i.e., if they are present)
    result = UINT16_Marshal(&result, buffer, size);
    // Marshal the structure
    result =
        (UINT16)(result
                 + TPMS_CREATION_DATA_Marshal(
                     (TPMS_CREATION_DATA*)&(source->creationData), buffer, size));
    // if a buffer was provided, go back and fill in the actual size
    if(buffer != NULL)
        UINT16_TO_BYTE_ARRAY((result - 2), (*buffer - result));
    return result;
}

// Table "Definition of TPM_AT Constants" (Part 2: Structures)
TPM_RC
TPM_AT_Unmarshal(TPM_AT* target, BYTE** buffer, INT32* size)
{
```

```
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32*)target, buffer, size);
    if(result == TPM_RC_SUCCESS)
    {
        switch(*target)
        {
            case TPM_AT_ANY:
            case TPM_AT_ERROR:
            case TPM_AT_PV1:
            case TPM_AT_VEND:
                break;
            default:
                result = TPM_RC_VALUE;
                break;
        }
    }
    return result;
}
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_AT_Marshal(TPM_AT* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPM_AE Constants" (Part 2: Structures)
#  if !USE_MARSHALING_DEFINES
UINT16
TPM_AE_Marshal(TPM_AE* source, BYTE** buffer, INT32* size)
{
    return UINT32_Marshal((UINT32*)source, buffer, size);
}
#  endif  // !USE_MARSHALING_DEFINES

// Table "Definition of TPMS_AC_OUTPUT Structure" (Part 2: Structures)
UINT16
TPMS_AC_OUTPUT_Marshal(TPMS_AC_OUTPUT* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result = (UINT16)(result + TPM_AT_Marshal((TPM_AT*)&(source->tag), buffer, size));
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->data), buffer, size));
    return result;
}

// Table "Definition of TPML_AC_CAPABILITIES Structure" (Part 2: Structures)
UINT16
TPML_AC_CAPABILITIES_Marshal(TPML_AC_CAPABILITIES* source, BYTE** buffer, INT32* size)
{
    UINT16 result = 0;
    result =
        (UINT16)(result + UINT32_Marshal((UINT32*)&(source->count), buffer, size));
    result = (UINT16)(result
                      + TPMS_AC_OUTPUT_Array_Marshal(
                          (TPMS_AC_OUTPUT*)&(source->acCapabilities),
                          buffer,
                          size,
                          (INT32)source->count));
    return result;
}

// For structures that unmarshals/marshals an array, the code calls an
// un/marshaling function to process the array of the defined type.
// This section contains the functions that perform that operation
// Array Unmarshal/Marshal for BYTE
```

```
TPM_RC
BYTE_Array_Unmarshal(BYTE* target, BYTE** buffer, INT32* size, INT32 count)
{
    if(*size < count)
        return TPM_RC_INSUFFICIENT;
    memcpy(target, *buffer, count);
    *size -= count;
    *buffer += count;
    return TPM_RC_SUCCESS;
}
UINT16
BYTE_Array_Marshal(BYTE* source, BYTE** buffer, INT32* size, INT32 count)
{
    if(buffer != 0)
    {
        if((size == 0) || ((*size -= count) >= 0))
        {
            memcpy(*buffer, source, count);
            *buffer += count;
        }
        pAssert((size == 0) || (*size >= 0));
    }
    pAssert(count < INT16_MAX);
    return ((UINT16)count);
}

// Array Unmarshal and Marshal for TPM_ALG_ID
TPM_RC
TPM_ALG_ID_Array_Unmarshal(
    TPM_ALG_ID* target, BYTE** buffer, INT32* size, INT32 count)
{
    TPM_RC result = TPM_RC_SUCCESS;
    INT32  i;
    for(i = 0; ((result == TPM_RC_SUCCESS) && (i < count)); i++)
    {
        result = TPM_ALG_ID_Unmarshal(&target[i], buffer, size);
    }
    return result;
}
UINT16
TPM_ALG_ID_Array_Marshal(TPM_ALG_ID* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPM_ALG_ID_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Unmarshal and Marshal for TPM_CC
TPM_RC
TPM_CC_Array_Unmarshal(TPM_CC* target, BYTE** buffer, INT32* size, INT32 count)
{
    TPM_RC result = TPM_RC_SUCCESS;
    INT32  i;
    for(i = 0; ((result == TPM_RC_SUCCESS) && (i < count)); i++)
    {
        result = TPM_CC_Unmarshal(&target[i], buffer, size);
    }
    return result;
}
UINT16
TPM_CC_Array_Marshal(TPM_CC* source, BYTE** buffer, INT32* size, INT32 count)
{
```

```
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPM_CC_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Marshal for TPM_ECC_CURVE
#  if ALG_ECC
UINT16
TPM_ECC_CURVE_Array_Marshal(
    TPM_ECC_CURVE* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPM_ECC_CURVE_Marshal(&source[i], buffer, size);
    }
    return result;
}
#  endif  // ALG_ECC

// Array Marshal for TPM_HANDLE
UINT16
TPM_HANDLE_Array_Marshal(TPM_HANDLE* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPM_HANDLE_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Unmarshal and Marshal for TPM2B_DIGEST
TPM_RC
TPM2B_DIGEST_Array_Unmarshal(
    TPM2B_DIGEST* target, BYTE** buffer, INT32* size, INT32 count)
{
    TPM_RC result = TPM_RC_SUCCESS;
    INT32  i;
    for(i = 0; ((result == TPM_RC_SUCCESS) && (i < count)); i++)
    {
        result = TPM2B_DIGEST_Unmarshal(&target[i], buffer, size);
    }
    return result;
}
UINT16
TPM2B_DIGEST_Array_Marshal(
    TPM2B_DIGEST* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPM2B_DIGEST_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Unmarshal and Marshal for TPM2B_VENDOR_PROPERTY
TPM_RC
```

```
TPM2B_VENDOR_PROPERTY_Array_Unmarshal(
    TPM2B_VENDOR_PROPERTY* target, BYTE** buffer, INT32* size, INT32 count)
{
    TPM_RC result = TPM_RC_SUCCESS;
    INT32  i;
    for(i = 0; ((result == TPM_RC_SUCCESS) && (i < count)); i++)
    {
        result = TPM2B_VENDOR_PROPERTY_Unmarshal(&target[i], buffer, size);
    }
    return result;
}
UINT16
TPM2B_VENDOR_PROPERTY_Array_Marshal(
    TPM2B_VENDOR_PROPERTY* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPM2B_VENDOR_PROPERTY_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Marshal for TPMA_CC
UINT16
TPMA_CC_Array_Marshal(TPMA_CC* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMA_CC_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Marshal for TPMS_AC_OUTPUT
UINT16
TPMS_AC_OUTPUT_Array_Marshal(
    TPMS_AC_OUTPUT* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMS_AC_OUTPUT_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Marshal for TPMS_ACT_DATA
UINT16
TPMS_ACT_DATA_Array_Marshal(
    TPMS_ACT_DATA* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMS_ACT_DATA_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Marshal for TPMS_ALG_PROPERTY
```

```
UINT16
TPMS_ALG_PROPERTY_Array_Marshal(
    TPMS_ALG_PROPERTY* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMS_ALG_PROPERTY_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Unmarshal and Marshal for TPMS_PCR_SELECTION
TPM_RC
TPMS_PCR_SELECTION_Array_Unmarshal(
    TPMS_PCR_SELECTION* target, BYTE** buffer, INT32* size, INT32 count)
{
    TPM_RC result = TPM_RC_SUCCESS;
    INT32  i;
    for(i = 0; ((result == TPM_RC_SUCCESS) && (i < count)); i++)
    {
        result = TPMS_PCR_SELECTION_Unmarshal(&target[i], buffer, size);
    }
    return result;
}
UINT16
TPMS_PCR_SELECTION_Array_Marshal(
    TPMS_PCR_SELECTION* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMS_PCR_SELECTION_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Marshal for TPMS_TAGGED_PCR_SELECT
UINT16
TPMS_TAGGED_PCR_SELECT_Array_Marshal(
    TPMS_TAGGED_PCR_SELECT* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMS_TAGGED_PCR_SELECT_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Marshal for TPMS_TAGGED_POLICY
UINT16
TPMS_TAGGED_POLICY_Array_Marshal(
    TPMS_TAGGED_POLICY* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMS_TAGGED_POLICY_Marshal(&source[i], buffer, size);
    }
    return result;
}
```

```
// Array Marshal for TPMS_TAGGED_PROPERTY
UINT16
TPMS_TAGGED_PROPERTY_Array_Marshal(
    TPMS_TAGGED_PROPERTY* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMS_TAGGED_PROPERTY_Marshal(&source[i], buffer, size);
    }
    return result;
}

// Array Unmarshal and Marshal for TPMT_HA
TPM_RC
TPMT_HA_Array_Unmarshal(
    TPMT_HA* target, BYTE** buffer, INT32* size, BOOL flag, INT32 count)
{
    TPM_RC result = TPM_RC_SUCCESS;
    INT32  i;
    for(i = 0; ((result == TPM_RC_SUCCESS) && (i < count)); i++)
    {
        result = TPMT_HA_Unmarshal(&target[i], buffer, size, flag);
    }
    return result;
}
UINT16
TPMT_HA_Array_Marshal(TPMT_HA* source, BYTE** buffer, INT32* size, INT32 count)
{
    UINT16 result = 0;
    INT32  i;
    for(i = 0; i < count; i++)
    {
        result += TPMT_HA_Marshal(&source[i], buffer, size);
    }
    return result;
}
#endif  // !TABLE_DRIVEN_MARSHAL
```

7.190 /tpm/src/support/MathOnByteBuffers.c

```
//** Introduction
//
// This file contains implementation of the math functions that are performed
// with canonical integers in byte buffers. The canonical integer is
// big-endian bytes.
//
#include "Tpm.h"
#include "TpmMath_Util_fp.h"

//** Functions

//*** UnsignedCmpB
// This function compare two unsigned values. The values are byte-aligned,
// big-endian numbers (e.g, a hash).
//  Return Type: int
//      1           if (a > b)
//      0           if (a = b)
//      -1          if (a < b)
LIB_EXPORT int UnsignedCompareB(UINT32      aSize,  // IN: size of a
                                const BYTE* a,      // IN: a
                                UINT32      bSize,  // IN: size of b
                                const BYTE* b       // IN: b
```

```
)
{
    UINT32 i;
    if(aSize > bSize)
        return 1;
    else if(aSize < bSize)
        return -1;
    else
    {
        for(i = 0; i < aSize; i++)
        {
            if(a[i] != b[i])
                return (a[i] > b[i]) ? 1 : -1;
        }
    }
    // Will return == if sizes are both zero
    return 0;
}

//***SignedCompareB()
// Compare two signed integers:
//  Return Type: int
//      1         if a > b
//      0         if a = b
//      -1        if a < b
int SignedCompareB(const UINT32 aSize,  // IN: size of a
                   const BYTE*  a,      // IN: a buffer
                   const UINT32 bSize,  // IN: size of b
                   const BYTE*  b       // IN: b buffer
)
{
    // are the signs different ?
    if(((a[0] ^ b[0]) & 0x80) > 0)
        // if the signs are different, then a is less than b if a is negative.
        return a[0] & 0x80 ? -1 : 1;
    else
        // do unsigned compare function
        return UnsignedCompareB(aSize, a, bSize, b);
}

#if ALG_RSA
//*** ModExpB
// This function is used to do modular exponentiation in support of RSA.
// The most typical uses are: 'c' = 'm'^'e' mod 'n' (RSA encrypt) and
// 'm' = 'c'^'d' mod 'n' (RSA decrypt).  When doing decryption, the 'e' parameter
// of the function will contain the private exponent 'd' instead of the public
// exponent 'e'.
//
// If the results will not fit in the provided buffer,
// an error is returned (CRYPT_ERROR_UNDERFLOW). If the results is smaller
// than the buffer, the results is de-normalized.
//
// This version is intended for use with RSA and requires that 'm' be
// less than 'n'.
//
//  Return Type: TPM_RC
//      TPM_RC_SIZE         number to exponentiate is larger than the modulus
//      TPM_RC_NO_RESULT    result will not fit into the provided buffer
//
TPM_RC
ModExpB(UINT32 cSize,  // IN: the size of the output buffer. It will
                       //     need to be the same size as the modulus
        BYTE* c,       // OUT: the buffer to receive the results
                       //     (c->size must be set to the maximum size
                       //     for the returned value)
        const UINT32 mSize,
```

```
        const BYTE*  m,  // IN: number to exponentiate
        const UINT32 eSize,
        const BYTE*  e,  // IN: power
        const UINT32 nSize,
        const BYTE*  n  // IN: modulus
)
{
    CRYPT_INT_MAX(bnC);
    CRYPT_INT_MAX(bnM);
    CRYPT_INT_MAX(bnE);
    CRYPT_INT_MAX(bnN);
    NUMBYTES tSize  = (NUMBYTES)nSize;
    TPM_RC   retVal = TPM_RC_SUCCESS;

    // Convert input parameters
    ExtMath_IntFromBytes(bnM, m, (NUMBYTES)mSize);
    ExtMath_IntFromBytes(bnE, e, (NUMBYTES)eSize);
    ExtMath_IntFromBytes(bnN, n, (NUMBYTES)nSize);

    // Make sure that the output is big enough to hold the result
    // and that 'm' is less than 'n' (the modulus)
    if(cSize < nSize)
        ERROR_EXIT(TPM_RC_NO_RESULT);
    if(ExtMath_UnsignedCmp(bnM, bnN) >= 0)
        ERROR_EXIT(TPM_RC_SIZE);
    ExtMath_ModExp(bnC, bnM, bnE, bnN);
    ExtMath_IntToBytes(bnC, c, &tSize);
Exit:
    return retVal;
}
#endif  // ALG_RSA

//*** DivideB()
// Divide an integer ('n') by an integer ('d') producing a quotient ('q') and
// a remainder ('r'). If 'q' or 'r' is not needed, then the pointer to them
// may be set to NULL.
//
//  Return Type: TPM_RC
//      TPM_RC_NO_RESULT         'q' or 'r' is too small to receive the result
//
LIB_EXPORT TPM_RC DivideB(const TPM2B* n,  // IN: numerator
                          const TPM2B* d,  // IN: denominator
                          TPM2B*       q,  // OUT: quotient
                          TPM2B*       r   // OUT: remainder
)
{
    CRYPT_INT_MAX_INITIALIZED(bnN, n);
    CRYPT_INT_MAX_INITIALIZED(bnD, d);
    CRYPT_INT_MAX(bnQ);
    CRYPT_INT_MAX(bnR);
    //
    // Do divide with converted values
    ExtMath_Divide(bnQ, bnR, bnN, bnD);

    // Convert the Crypt_Int* result back to 2B format using the size of the original
    // number
    if(q != NULL)
        if(!TpmMath_IntTo2B(bnQ, q, q->size))
            return TPM_RC_NO_RESULT;
    if(r != NULL)
        if(!TpmMath_IntTo2B(bnR, r, r->size))
            return TPM_RC_NO_RESULT;
    return TPM_RC_SUCCESS;
}

//*** AdjustNumberB()
```

```
// Remove/add leading zeros from a number in a TPM2B. Will try to make the number
// by adding or removing leading zeros. If the number is larger than the requested
// size, it will make the number as small as possible. Setting 'requestedSize' to
// zero is equivalent to requesting that the number be normalized.
UINT16
AdjustNumberB(TPM2B* num, UINT16 requestedSize)
{
    BYTE*  from;
    UINT16 i;
    // See if number is already the requested size
    if(num->size == requestedSize)
        return requestedSize;
    from = num->buffer;
    if(num->size > requestedSize)
    {
        // This is a request to shift the number to the left (remove leading zeros)
        // Find the first non-zero byte. Don't look past the point where removing
        // more zeros would make the number smaller than requested, and don't throw
        // away any significant digits.
        for(i = num->size; *from == 0 && i > requestedSize; from++, i--)
            ;
        if(i < num->size)
        {
            num->size = i;
            MemoryCopy(num->buffer, from, i);
        }
    }
    // This is a request to shift the number to the right (add leading zeros)
    else
    {
        MemoryCopy(&num->buffer[requestedSize - num->size], num->buffer, num->size);
        MemorySet(num->buffer, 0, requestedSize - num->size);
        num->size = requestedSize;
    }
    return num->size;
}

//*** ShiftLeft()
// This function shifts a byte buffer (a TPM2B) one byte to the left. That is,
// the most significant bit of the most significant byte is lost.
TPM2B* ShiftLeft(TPM2B* value  // IN/OUT: value to shift and shifted value out
)
{
    UINT16 count  = value->size;
    BYTE*  buffer = value->buffer;
    if(count > 0)
    {
        for(count -= 1; count > 0; buffer++, count--)
        {
            buffer[0] = (buffer[0] << 1) + ((buffer[1] & 0x80) ? 1 : 0);
        }
        *buffer <<= 1;
    }
    return value;
}
```

7.191 /tpm/src/support/Memory.c

```
//** Description
// This file contains a set of miscellaneous memory manipulation routines. Many
// of the functions have the same semantics as functions defined in string.h.
// Those functions are not used directly in the TPM because they are not 'safe'
//
// This version uses string.h after adding guards.  This is because the math
// libraries invariably use those functions so it is not practical to prevent
```

```
// those library functions from being pulled into the build.

//** Includes and Data Definitions
#include "Tpm.h"
#include "Memory_fp.h"

//** Functions

//*** MemoryCopy()
// This is an alias for memmove. This is used in place of memcpy because
// some of the moves may overlap and rather than try to make sure that
// memmove is used when necessary, it is always used.
void MemoryCopy(void* dest, const void* src, int sSize)
{
    if(dest != src)
        memmove(dest, src, sSize);
}

//*** MemoryEqual()
// This function indicates if two buffers have the same values in the indicated
// number of bytes.
//  Return Type: BOOL
//      TRUE(1)         all octets are the same
//      FALSE(0)        all octets are not the same
BOOL MemoryEqual(const void*  buffer1,  // IN: compare buffer1
                 const void*  buffer2,  // IN: compare buffer2
                 unsigned int size      // IN: size of bytes being compared
)
{
    BYTE        equal = 0;
    const BYTE* b1    = (BYTE*)buffer1;
    const BYTE* b2    = (BYTE*)buffer2;
    //
    // Compare all bytes so that there is no leakage of information
    // due to timing differences.
    for(; size > 0; size--)
        equal |= (*b1++ ^ *b2++);
    return (equal == 0);
}

//*** MemoryCopy2B()
// This function copies a TPM2B. This can be used when the TPM2B types are
// the same or different.
//
// This function returns the number of octets in the data buffer of the TPM2B.
LIB_EXPORT INT16 MemoryCopy2B(TPM2B*       dest,    // OUT: receiving TPM2B
                              const TPM2B* source,  // IN: source TPM2B
                              unsigned int dSize  // IN: size of the receiving buffer
)
{
    pAssert(dest != NULL);
    if(source == NULL)
        dest->size = 0;
    else
    {
        pAssert(source->size <= dSize);
        MemoryCopy(dest->buffer, source->buffer, source->size);
        dest->size = source->size;
    }
    return dest->size;
}

//*** MemoryConcat2B()
// This function will concatenate the buffer contents of a TPM2B to
// the buffer contents of another TPM2B and adjust the size accordingly
//      ('a' := ('a' | 'b')).
```

```
void MemoryConcat2B(
    TPM2B*       aInOut,    // IN/OUT: destination 2B
    TPM2B*       bIn,       // IN: second 2B
    unsigned int aMaxSize   // IN: The size of aInOut.buffer (max values for
                            //     aInOut.size)
)
{
    pAssert(bIn->size <= aMaxSize - aInOut->size);
    MemoryCopy(&aInOut->buffer[aInOut->size], &bIn->buffer, bIn->size);
    aInOut->size = aInOut->size + bIn->size;
    return;
}

//*** MemoryEqual2B()
// This function will compare two TPM2B structures. To be equal, they
// need to be the same size and the buffer contexts need to be the same
// in all octets.
//  Return Type: BOOL
//      TRUE(1)         size and buffer contents are the same
//      FALSE(0)        size or buffer contents are not the same
BOOL MemoryEqual2B(const TPM2B* aIn,  // IN: compare value
                   const TPM2B* bIn   // IN: compare value
)
{
    if(aIn->size != bIn->size)
        return FALSE;
    return MemoryEqual(aIn->buffer, bIn->buffer, aIn->size);
}

//*** MemorySet()
// This function will set all the octets in the specified memory range to
// the specified octet value.
// Note: A previous version had an additional parameter (dSize) that was
// intended to make sure that the destination would not be overrun. The
// problem is that, in use, all that was happening was that the value of
// size was used for dSize so there was no benefit in the extra parameter.
void MemorySet(void* dest, int value, size_t size)
{
    memset(dest, value, size);
}

//*** MemoryPad2B()
// Function to pad a TPM2B with zeros and adjust the size.
void MemoryPad2B(TPM2B* b, UINT16 newSize)
{
    MemorySet(&b->buffer[b->size], 0, newSize - b->size);
    b->size = newSize;
}

//*** Uint16ToByteArray()
// Function to write an integer to a byte array
void Uint16ToByteArray(UINT16 i, BYTE* a)
{
    a[1] = (BYTE)(i);
    i >>= 8;
    a[0] = (BYTE)(i);
}

//*** Uint32ToByteArray()
// Function to write an integer to a byte array
void Uint32ToByteArray(UINT32 i, BYTE* a)
{
    a[3] = (BYTE)(i);
    i >>= 8;
    a[2] = (BYTE)(i);
    i >>= 8;
```

```
    a[1] = (BYTE)(i);
    i >>= 8;
    a[0] = (BYTE)(i);
}

//*** Uint64ToByteArray()
// Function to write an integer to a byte array
void Uint64ToByteArray(UINT64 i, BYTE* a)
{
    a[7] = (BYTE)(i);
    i >>= 8;
    a[6] = (BYTE)(i);
    i >>= 8;
    a[5] = (BYTE)(i);
    i >>= 8;
    a[4] = (BYTE)(i);
    i >>= 8;
    a[3] = (BYTE)(i);
    i >>= 8;
    a[2] = (BYTE)(i);
    i >>= 8;
    a[1] = (BYTE)(i);
    i >>= 8;
    a[0] = (BYTE)(i);
}

//*** ByteArrayToUint8()
// Function to write a UINT8 to a byte array. This is included for completeness
// and to allow certain macro expansions
UINT8
ByteArrayToUint8(BYTE* a)
{
    return *a;
}

//*** ByteArrayToUint16()
// Function to write an integer to a byte array
UINT16
ByteArrayToUint16(BYTE* a)
{
    return ((UINT16)a[0] << 8) + a[1];
}

//*** ByteArrayToUint32()
// Function to write an integer to a byte array
UINT32
ByteArrayToUint32(BYTE* a)
{
    return (UINT32)((((((UINT32)a[0] << 8) + a[1]) << 8) + (UINT32)a[2]) << 8) + a[3];
}

//*** ByteArrayToUint64()
// Function to write an integer to a byte array
UINT64
ByteArrayToUint64(BYTE* a)
{
    return (((UINT64)BYTE_ARRAY_TO_UINT32(a)) << 32) + BYTE_ARRAY_TO_UINT32(&a[4]);
}
```

7.192 /tpm/src/support/Power.c

```
//** Description

// This file contains functions that receive the simulated power state
// transitions of the TPM.
```

```
//** Includes and Data Definitions
#define POWER_C
#include "Tpm.h"

//** Functions

//*** TPMInit()
// This function is used to process a power on event.
void TPMInit(void)
{
    // Set state as not initialized. This means that Startup is required
    g_initialized = FALSE;
    return;
}

//*** TPMRegisterStartup()
// This function registers the fact that the TPM has been initialized
// (a TPM2_Startup() has completed successfully).
BOOL TPMRegisterStartup(void)
{
    g_initialized = TRUE;
    return TRUE;
}

//*** TPMIsStarted()
// Indicates if the TPM has been initialized (a TPM2_Startup() has completed
// successfully after a _TPM_Init).
//  Return Type: BOOL
//      TRUE(1)         TPM has been initialized
//      FALSE(0)        TPM has not been initialized
BOOL TPMIsStarted(void)
{
    return g_initialized;
}
```

7.193 /tpm/src/support/PropertyCap.c

```
//** Description
// This file contains the functions that are used for accessing the
// TPM_CAP_TPM_PROPERTY values.

//** Includes

#include "Tpm.h"

//** Functions

//*** TPMPropertyIsDefined()
// This function accepts a property selection and, if so, sets 'value'
// to the value of the property.
//
// All the fixed values are vendor dependent or determined by a
// platform-specific specification. The values in the table below
// are examples and should be changed by the vendor.
//  Return Type: BOOL
//      TRUE(1)         referenced property exists and 'value' set
//      FALSE(0)        referenced property does not exist
static BOOL TPMPropertyIsDefined(TPM_PT  property,  // IN: property
                                 UINT32* value      // OUT: property value
)
{
    switch(property)
    {
        case TPM_PT_FAMILY_INDICATOR:
```

```
        // from the title page of the specification
        // For this specification, the value is "2.0".
        *value = TPM_SPEC_FAMILY;
        break;
    case TPM_PT_LEVEL:
        // from the title page of the specification
        *value = TPM_SPEC_LEVEL;
        break;
    case TPM_PT_REVISION:
        // from the title page of the specification
        *value = TPM_SPEC_VERSION;
        break;
    case TPM_PT_DAY_OF_YEAR:
        // computed from the date value on the title page of the specification
        *value = TPM_SPEC_DAY_OF_YEAR;
        break;
    case TPM_PT_YEAR:
        // from the title page of the specification
        *value = TPM_SPEC_YEAR;
        break;

    case TPM_PT_MANUFACTURER:
        // the vendor ID unique to each TPM manufacturer
        *value = _plat__GetManufacturerCapabilityCode();
        break;

    case TPM_PT_VENDOR_STRING_1:
        // the first four characters of the vendor ID string
        *value = _plat__GetVendorCapabilityCode(1);
        break;

    case TPM_PT_VENDOR_STRING_2:
        // the second four characters of the vendor ID string
        *value = _plat__GetVendorCapabilityCode(2);
        break;

    case TPM_PT_VENDOR_STRING_3:
        // the third four characters of the vendor ID string
        *value = _plat__GetVendorCapabilityCode(3);
        break;

    case TPM_PT_VENDOR_STRING_4:
        // the fourth four characters of the vendor ID string
        *value = _plat__GetVendorCapabilityCode(4);
        break;

    case TPM_PT_VENDOR_TPM_TYPE:
        // vendor-defined value indicating the TPM model
        // We just make up a number here
        *value = _plat__GetTpmType();
        break;

    case TPM_PT_FIRMWARE_VERSION_1:
        // more significant 32-bits of a vendor-specific value
        *value = gp.firmwareV1;
        break;
    case TPM_PT_FIRMWARE_VERSION_2:
        // less significant 32-bits of a vendor-specific value
        *value = gp.firmwareV2;
        break;
    case TPM_PT_INPUT_BUFFER:
        // maximum size of TPM2B_MAX_BUFFER
        *value = MAX_DIGEST_BUFFER;
        break;
    case TPM_PT_HR_TRANSIENT_MIN:
        // minimum number of transient objects that can be held in TPM
```

```
            // RAM
            *value = MAX_LOADED_OBJECTS;
            break;
        case TPM_PT_HR_PERSISTENT_MIN:
            // minimum number of persistent objects that can be held in
            // TPM NV memory
            // In this implementation, there is no minimum number of
            // persistent objects.
            *value = MIN_EVICT_OBJECTS;
            break;
        case TPM_PT_HR_LOADED_MIN:
            // minimum number of authorization sessions that can be held in
            // TPM RAM
            *value = MAX_LOADED_SESSIONS;
            break;
        case TPM_PT_ACTIVE_SESSIONS_MAX:
            // number of authorization sessions that may be active at a time
            *value = MAX_ACTIVE_SESSIONS;
            break;
        case TPM_PT_PCR_COUNT:
            // number of PCR implemented
            *value = IMPLEMENTATION_PCR;
            break;
        case TPM_PT_PCR_SELECT_MIN:
            // minimum number of bytes in a TPMS_PCR_SELECT.sizeOfSelect
            *value = PCR_SELECT_MIN;
            break;
        case TPM_PT_CONTEXT_GAP_MAX:
            // maximum allowed difference (unsigned) between the contextID
            // values of two saved session contexts
            *value = ((UINT32)1 << (sizeof(CONTEXT_SLOT) * 8)) - 1;
            break;
        case TPM_PT_NV_COUNTERS_MAX:
            // maximum number of NV indexes that are allowed to have the
            // TPMA_NV_COUNTER attribute SET
            // In this implementation, there is no limitation on the number
            // of counters, except for the size of the NV Index memory.
            *value = 0;
            break;
        case TPM_PT_NV_INDEX_MAX:
            // maximum size of an NV index data area
            *value = MAX_NV_INDEX_SIZE;
            break;
        case TPM_PT_MEMORY:
            // a TPMA_MEMORY indicating the memory management method for the TPM
            {
                union
                {
                    TPMA_MEMORY att;
                    UINT32      u32;
                } attributes = {TPMA_ZERO_INITIALIZER()};
                SET_ATTRIBUTE(attributes.att, TPMA_MEMORY, sharedNV);
                SET_ATTRIBUTE(attributes.att, TPMA_MEMORY, objectCopiedToRam);

                // Note: For a LSb0 machine, the bits in a bit field are in the
correct
                // order even if the machine is MSB0. For a MSb0 machine, a TPMA will
                // be an integer manipulated by masking (USE_BIT_FIELD_STRUCTURES will
                // be NO) so the bits are manipulate correctly.
                *value = attributes.u32;
                break;
            }
        case TPM_PT_CLOCK_UPDATE:
            // interval, in seconds, between updates to the copy of
            // TPMS_TIME_INFO .clock in NV
            *value = (1 << NV_CLOCK_UPDATE_INTERVAL);
```

```
            break;
        case TPM_PT_CONTEXT_HASH:
            // algorithm used for the integrity hash on saved contexts and
            // for digesting the fuData of TPM2_FirmwareRead()
            *value = CONTEXT_INTEGRITY_HASH_ALG;
            break;
        case TPM_PT_CONTEXT_SYM:
            // algorithm used for encryption of saved contexts
            *value = CONTEXT_ENCRYPT_ALG;
            break;
        case TPM_PT_CONTEXT_SYM_SIZE:
            // size of the key used for encryption of saved contexts
            *value = CONTEXT_ENCRYPT_KEY_BITS;
            break;
        case TPM_PT_ORDERLY_COUNT:
            // maximum difference between the volatile and non-volatile
            // versions of TPMA_NV_COUNTER that have TPMA_NV_ORDERLY SET
            *value = MAX_ORDERLY_COUNT;
            break;
        case TPM_PT_MAX_COMMAND_SIZE:
            // maximum value for 'commandSize'
            *value = MAX_COMMAND_SIZE;
            break;
        case TPM_PT_MAX_RESPONSE_SIZE:
            // maximum value for 'responseSize'
            *value = MAX_RESPONSE_SIZE;
            break;
        case TPM_PT_MAX_DIGEST:
            // maximum size of a digest that can be produced by the TPM
            *value = sizeof(TPMU_HA);
            break;
        case TPM_PT_MAX_OBJECT_CONTEXT:
// Header has 'sequence', 'handle' and 'hierarchy'
#define SIZE_OF_CONTEXT_HEADER \
    sizeof(UINT64) + sizeof(TPMI_DH_CONTEXT) + sizeof(TPMI_RH_HIERARCHY)
#define SIZE_OF_CONTEXT_INTEGRITY (sizeof(UINT16) + CONTEXT_INTEGRITY_HASH_SIZE)
#define SIZE_OF_FINGERPRINT       sizeof(UINT64)
#define SIZE_OF_CONTEXT_BLOB_OVERHEAD \
    (sizeof(UINT16) + SIZE_OF_CONTEXT_INTEGRITY + SIZE_OF_FINGERPRINT)
#define SIZE_OF_CONTEXT_OVERHEAD \
    (SIZE_OF_CONTEXT_HEADER + SIZE_OF_CONTEXT_BLOB_OVERHEAD)
#if 0
            // maximum size of a TPMS_CONTEXT that will be returned by
            // TPM2_ContextSave for object context
            *value = 0;
            // adding sequence, saved handle and hierarchy
            *value += sizeof(UINT64) + sizeof(TPMI_DH_CONTEXT) +
                sizeof(TPMI_RH_HIERARCHY);
            // add size field in TPM2B_CONTEXT
            *value += sizeof(UINT16);
            // add integrity hash size
            *value += sizeof(UINT16) +
                CryptHashGetDigestSize(CONTEXT_INTEGRITY_HASH_ALG);
            // Add fingerprint size, which is the same as sequence size
            *value += sizeof(UINT64);
            // Add OBJECT structure size
            *value += sizeof(OBJECT);
#else
            // the maximum size of a TPMS_CONTEXT that will be returned by
            // TPM2_ContextSave for object context
            *value = SIZE_OF_CONTEXT_OVERHEAD + sizeof(OBJECT);
#endif
            break;
        case TPM_PT_MAX_SESSION_CONTEXT:
#if 0
```

```
            // the maximum size of a TPMS_CONTEXT that will be returned by
            // TPM2_ContextSave for object context
            *value = 0;
            // adding sequence, saved handle and hierarchy
            *value += sizeof(UINT64) + sizeof(TPMI_DH_CONTEXT) +
                sizeof(TPMI_RH_HIERARCHY);
            // Add size field in TPM2B_CONTEXT
            *value += sizeof(UINT16);
// Add integrity hash size
            *value += sizeof(UINT16) +
                CryptHashGetDigestSize(CONTEXT_INTEGRITY_HASH_ALG);
      // Add fingerprint size, which is the same as sequence size
            *value += sizeof(UINT64);
            // Add SESSION structure size
            *value += sizeof(SESSION);
#else
            // the maximum size of a TPMS_CONTEXT that will be returned by
            // TPM2_ContextSave for object context
            *value = SIZE_OF_CONTEXT_OVERHEAD + sizeof(SESSION);
#endif
            break;
        case TPM_PT_PS_FAMILY_INDICATOR:
            // platform specific values for the TPM_PT_PS parameters from
            // the relevant platform-specific specification
            // In this reference implementation, all of these values are 0.
            *value = PLATFORM_FAMILY;
            break;
        case TPM_PT_PS_LEVEL:
            // level of the platform-specific specification
            *value = PLATFORM_LEVEL;
            break;
        case TPM_PT_PS_REVISION:
            // specification Revision times 100 for the platform-specific
            // specification
            *value = PLATFORM_VERSION;
            break;
        case TPM_PT_PS_DAY_OF_YEAR:
            // platform-specific specification day of year using TCG calendar
            *value = PLATFORM_DAY_OF_YEAR;
            break;
        case TPM_PT_PS_YEAR:
            // platform-specific specification year using the CE
            *value = PLATFORM_YEAR;
            break;
        case TPM_PT_SPLIT_MAX:
            // number of split signing operations supported by the TPM
            *value = 0;
#if ALG_ECC
            *value = sizeof(gr.commitArray) * 8;
#endif
            break;
        case TPM_PT_TOTAL_COMMANDS:
            // total number of commands implemented in the TPM
            // Since the reference implementation does not have any
            // vendor-defined commands, this will be the same as the
            // number of library commands.
            {
#if COMPRESSED_LISTS
                (*value) = COMMAND_COUNT;
#else
                COMMAND_INDEX commandIndex;
                *value = 0;

                // scan all implemented commands
                for(commandIndex = GetClosestCommandIndex(0);
                    commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
```

```
                    commandIndex = GetNextCommandIndex(commandIndex))
                {
                    (*value)++;  // count of all implemented
                }
#endif
                break;
            }
        case TPM_PT_LIBRARY_COMMANDS:
            // number of commands from the TPM library that are implemented
            {
#if COMPRESSED_LISTS
                *value = LIBRARY_COMMAND_ARRAY_SIZE;
#else
                COMMAND_INDEX commandIndex;
                *value = 0;

                // scan all implemented commands
                for(commandIndex = GetClosestCommandIndex(0);
                    commandIndex < LIBRARY_COMMAND_ARRAY_SIZE;
                    commandIndex = GetNextCommandIndex(commandIndex))
                {
                    (*value)++;
                }
#endif
                break;
            }
        case TPM_PT_VENDOR_COMMANDS:
            // number of vendor commands that are implemented
            *value = VENDOR_COMMAND_ARRAY_SIZE;
            break;
        case TPM_PT_NV_BUFFER_MAX:
            // Maximum data size in an NV write command
            *value = MAX_NV_BUFFER_SIZE;
            break;
        case TPM_PT_MODES:
        {
            union
            {
                TPMA_MODES attr;
                UINT32     u32;
            } flags = {TPMA_ZERO_INITIALIZER()};
#if FIPS_COMPLIANT
            SET_ATTRIBUTE(flags.attr, TPMA_MODES, FIPS_140_2);
#endif
            *value = flags.u32;
            break;
        }
        case TPM_PT_MAX_CAP_BUFFER:
            *value = MAX_CAP_BUFFER;
            break;
        case TPM_PT_FIRMWARE_SVN:
            *value = _plat__GetTpmFirmwareSvn();
            break;
        case TPM_PT_FIRMWARE_MAX_SVN:
            *value = _plat__GetTpmFirmwareMaxSvn();
            break;

        // Start of variable commands
        case TPM_PT_PERMANENT:
            // TPMA_PERMANENT
            {
                union
                {
                    TPMA_PERMANENT attr;
                    UINT32         u32;
                } flags = {TPMA_ZERO_INITIALIZER()};
```

```
                if(gp.ownerAuth.t.size != 0)
                    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, ownerAuthSet);
                if(gp.endorsementAuth.t.size != 0)
                    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, endorsementAuthSet);
                if(gp.lockoutAuth.t.size != 0)
                    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, lockoutAuthSet);
                if(gp.disableClear)
                    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, disableClear);
                if(gp.failedTries >= gp.maxTries)
                    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, inLockout);
                // In this implementation, EPS is always generated by TPM
                SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, tpmGeneratedEPS);

                // Note: For a LSb0 machine, the bits in a bit field are in the
correct
                // order even if the machine is MSB0. For a MSb0 machine, a TPMA will
                // be an integer manipulated by masking (USE_BIT_FIELD_STRUCTURES will
                // be NO) so the bits are manipulate correctly.
                *value = flags.u32;
                break;
            }
        case TPM_PT_STARTUP_CLEAR:
            // TPMA_STARTUP_CLEAR
            {
                union
                {
                    TPMA_STARTUP_CLEAR attr;
                    UINT32             u32;
                } flags = {TPMA_ZERO_INITIALIZER()};
                //
                if(g_phEnable)
                    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, phEnable);
                if(gc.shEnable)
                    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, shEnable);
                if(gc.ehEnable)
                    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, ehEnable);
                if(gc.phEnableNV)
                    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, phEnableNV);
                if(g_prevOrderlyState != SU_NONE_VALUE)
                    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, orderly);

                // Note: For a LSb0 machine, the bits in a bit field are in the
correct
                // order even if the machine is MSB0. For a MSb0 machine, a TPMA will
                // be an integer manipulated by masking (USE_BIT_FIELD_STRUCTURES will
                // be NO) so the bits are manipulate correctly.
                *value = flags.u32;
                break;
            }
        case TPM_PT_HR_NV_INDEX:
            // number of NV indexes currently defined
            *value = NvCapGetIndexNumber();
            break;
        case TPM_PT_HR_LOADED:
            // number of authorization sessions currently loaded into TPM
            // RAM
            *value = SessionCapGetLoadedNumber();
            break;
        case TPM_PT_HR_LOADED_AVAIL:
            // number of additional authorization sessions, of any type,
            // that could be loaded into TPM RAM
            *value = SessionCapGetLoadedAvail();
            break;
        case TPM_PT_HR_ACTIVE:
            // number of active authorization sessions currently being
            // tracked by the TPM
```

```
            *value = SessionCapGetActiveNumber();
            break;
        case TPM_PT_HR_ACTIVE_AVAIL:
            // number of additional authorization sessions, of any type,
            // that could be created
            *value = SessionCapGetActiveAvail();
            break;
        case TPM_PT_HR_TRANSIENT_AVAIL:
            // estimate of the number of additional transient objects that
            // could be loaded into TPM RAM
            *value = ObjectCapGetTransientAvail();
            break;
        case TPM_PT_HR_PERSISTENT:
            // number of persistent objects currently loaded into TPM
            // NV memory
            *value = NvCapGetPersistentNumber();
            break;
        case TPM_PT_HR_PERSISTENT_AVAIL:
            // number of additional persistent objects that could be loaded
            // into NV memory
            *value = NvCapGetPersistentAvail();
            break;
        case TPM_PT_NV_COUNTERS:
            // number of defined NV indexes that have NV TPMA_NV_COUNTER
            // attribute SET
            *value = NvCapGetCounterNumber();
            break;
        case TPM_PT_NV_COUNTERS_AVAIL:
            // number of additional NV indexes that can be defined with their
            // TPMA_NV_COUNTER attribute SET
            *value = NvCapGetCounterAvail();
            break;
        case TPM_PT_ALGORITHM_SET:
            // region code for the TPM
            *value = gp.algorithmSet;
            break;
        case TPM_PT_LOADED_CURVES:
#if ALG_ECC
            // number of loaded ECC curves
            *value = ECC_CURVE_COUNT;
#else   // ALG_ECC
            *value = 0;
#endif  // ALG_ECC
            break;
        case TPM_PT_LOCKOUT_COUNTER:
            // current value of the lockout counter
            *value = gp.failedTries;
            break;
        case TPM_PT_MAX_AUTH_FAIL:
            // number of authorization failures before DA lockout is invoked
            *value = gp.maxTries;
            break;
        case TPM_PT_LOCKOUT_INTERVAL:
            // number of seconds before the value reported by
            // TPM_PT_LOCKOUT_COUNTER is decremented
            *value = gp.recoveryTime;
            break;
        case TPM_PT_LOCKOUT_RECOVERY:
            // number of seconds after a lockoutAuth failure before use of
            // lockoutAuth may be attempted again
            *value = gp.lockoutRecovery;
            break;
        case TPM_PT_NV_WRITE_RECOVERY:
            // number of milliseconds before the TPM will accept another command
            // that will modify NV.
            // This should make a call to the platform code that is doing rate
```

```
            // limiting of NV. Rate limiting is not implemented in the reference
            // code so no call is made.
            *value = 0;
            break;
        case TPM_PT_AUDIT_COUNTER_0:
            // high-order 32 bits of the command audit counter
            *value = (UINT32)(gp.auditCounter >> 32);
            break;
        case TPM_PT_AUDIT_COUNTER_1:
            // low-order 32 bits of the command audit counter
            *value = (UINT32)(gp.auditCounter);
            break;
        default:
            // property is not defined
            return FALSE;
            break;
    }
    return TRUE;
}

//*** TPMCapGetProperties()
// This function is used to get the TPM_PT values. The search of properties will
// start at 'property' and continue until 'propertyList' has as many values as
// will fit, or the last property has been reported, or the list has as many
// values as requested in 'count'.
//  Return Type: TPMI_YES_NO
//  YES         more properties are available
//  NO          no more properties to be reported
TPMI_YES_NO
TPMCapGetProperties(TPM_PT property,  // IN: the starting TPM property
                    UINT32 count,     // IN: maximum number of returned
                                      //     properties
                    TPML_TAGGED_TPM_PROPERTY* propertyList  // OUT: property list
)
{
    TPMI_YES_NO more = NO;
    UINT32      i;
    UINT32      nextGroup;

    // initialize output property list
    propertyList->count = 0;

    // maximum count of properties we may return is MAX_PCR_PROPERTIES
    if(count > MAX_TPM_PROPERTIES)
        count = MAX_TPM_PROPERTIES;

    // if property is less than PT_FIXED, start from PT_FIXED
    if(property < PT_FIXED)
        property = PT_FIXED;
    // There is only the fixed and variable groups with the variable group coming
    // last
    if(property >= (PT_VAR + PT_GROUP))
        return more;

    // Don't read past the end of the selected group
    nextGroup = ((property / PT_GROUP) * PT_GROUP) + PT_GROUP;

    // Scan through the TPM properties of the requested group.
    for(i = property; i < nextGroup; i++)
    {
        UINT32 value;
        // if we have hit the end of the group, quit
        if(i != property && ((i % PT_GROUP) == 0))
            break;
        if(TPMPropertyIsDefined((TPM_PT)i, &value))
        {
```

```
            if(propertyList->count < count)
            {
                // If the list is not full, add this property
                propertyList->tpmProperty[propertyList->count].property = (TPM_PT)i;
                propertyList->tpmProperty[propertyList->count].value    = value;
                propertyList->count++;
            }
            else
            {
                // If the return list is full but there are more properties
                // available, set the indication and exit the loop.
                more = YES;
                break;
            }
        }
    }
    return more;
}

//*** TPMCapGetOneProperty()
// This function returns a single TPM property, if present.
BOOL TPMCapGetOneProperty(TPM_PT                pt,       // IN: the TPM property
                          TPMS_TAGGED_PROPERTY* property  // OUT: tagged property
)
{
    UINT32 value;

    if(TPMPropertyIsDefined((TPM_PT)pt, &value))
    {
        property->property = (TPM_PT)pt;
        property->value    = value;
        return TRUE;
    }

    return FALSE;
}
```

7.194 /tpm/src/support/Response.c

```
//** Description
// This file contains the common code for building a response header, including
// setting the size of the structure. 'command' may be NULL if result is
// not TPM_RC_SUCCESS.

//** Includes and Defines
#include "Tpm.h"
#include "Marshal.h"

//** BuildResponseHeader()
// Adds the response header to the response. It will update command->parameterSize
// to indicate the total size of the response.
void BuildResponseHeader(COMMAND* command,  // IN: main control structure
                         BYTE*    buffer,   // OUT: the output buffer
                         TPM_RC   result    // IN: the response code
)
{
    TPM_ST tag;
    UINT32 size;

    if(result != TPM_RC_SUCCESS)
    {
        tag  = TPM_ST_NO_SESSIONS;
        size = 10;
    }
    else
```

```
    {
        tag = command->tag;
        // Compute the overall size of the response
        size = STD_RESPONSE_HEADER + command->handleNum * sizeof(TPM_HANDLE);
        size += command->parameterSize;
        size += (command->tag == TPM_ST_SESSIONS) ? command->authSize + sizeof(UINT32)
                                                  : 0;
    }
    TPM_ST_Marshal(&tag, &buffer, NULL);
    UINT32_Marshal(&size, &buffer, NULL);
    TPM_RC_Marshal(&result, &buffer, NULL);
    if(result == TPM_RC_SUCCESS)
    {
        if(command->handleNum > 0)
            TPM_HANDLE_Marshal(&command->handles[0], &buffer, NULL);
        if(tag == TPM_ST_SESSIONS)
            UINT32_Marshal((UINT32*)&command->parameterSize, &buffer, NULL);
    }
    command->parameterSize = size;
}
```

7.195 /tpm/src/support/ResponseCodeProcessing.c

```
//** Description
// This file contains the miscellaneous functions for processing response codes.
// NOTE: Currently, there is only one.

//** Includes and Defines
#include "Tpm.h"

//** RcSafeAddToResult()
// Adds a modifier to a response code as long as the response code allows a modifier
// and no modifier has already been added.
TPM_RC
RcSafeAddToResult(TPM_RC responseCode, TPM_RC modifier)
{
    if((responseCode & RC_FMT1) && !(responseCode & 0xf40))
        return responseCode + modifier;
    else
        return responseCode;
}
```

7.196 /tpm/src/support/TableDrivenMarshal.c

```
#include <assert.h>

#include "Tpm.h"
#include "Marshal.h"
#include "TableMarshal.h"

#if TABLE_DRIVEN_MARSHAL

extern ArrayMarshal_mst ArrayLookupTable[];

extern UINT16           MarshalLookupTable[];

typedef struct
{
    int a;
} External_Structure_t;

extern struct Exernal_Structure_t MarshalData;

#  define IS_SUCCESS(UNMARSHAL_FUNCTION) \
```

```
      (TPM_RC_SUCCESS == (result = (UNMARSHAL_FUNCTION)))

marshalIndex_t IntegerDispatch[] = {UINT8_MARSHAL_REF,
                                    UINT16_MARSHAL_REF,
                                    UINT32_MARSHAL_REF,
                                    UINT64_MARSHAL_REF,
                                    INT8_MARSHAL_REF,
                                    INT16_MARSHAL_REF,
                                    INT32_MARSHAL_REF,
                                    INT64_MARSHAL_REF};

#  if 1
#    define GetDescriptor(reference) \
        ((MarshalHeader_mst*)(((BYTE*)(&MarshalData)) + (reference & NULL_MASK)))
#  else
static const MarshalHeader_mst* GetDescriptor(marshalIndex_t index)
{
    const MarshalHeader_mst* mst = MarshalLookupTable[index & NULL_MASK];
    return mst;
}
#  endif

#  define GetUnionDescriptor(_index_) ((UnionMarshal_mst*)GetDescriptor(_index_))
#  define GetArrayDescriptor(_index_) \
      ((ArrayMarshal_mst*))(ArrayLookupTable[_index_ & NULL_MASK])

//*** GetUnmarshaledInteger()
// Gets the unmarshaled value and normalizes it to a UIN32 for other
// processing (comparisons and such).
static UINT32 GetUnmarshaledInteger(marshalIndex_t type, const void* target)
{
    int size = (type & SIZE_MASK);
    //
    if(size == FOUR_BYTES)
        return *((UINT32*)target);
    if(type & IS_SIGNED)
    {
        if(size == TWO_BYTES)
            return (UINT32) * ((int16_t*)target);
        return (UINT32) * ((int8_t*)target);
    }
    if(size == TWO_BYTES)
        return (UINT32) * ((UINT16*)target);
    return (UINT32) * ((UINT8*)target);
}

static UINT32 GetSelector(void* structure, const UINT16* values, UINT16 descriptor)
{
    unsigned sel = GET_ELEMENT_NUMBER(descriptor);
    // Get the offset of the value in the unmarshaled structure
    const UINT16* entry = &values[(sel * 3)];
    //
    return GetUnmarshaledInteger(GET_ELEMENT_SIZE(entry[0]),
                                 ((UINT8*)structure) + entry[2]);
}

static TPM_RC UnmarshalBytes(
    UINT8*  target,  // IN/OUT: place to put the bytes
    UINT8** buffer,  // IN/OUT: source of the input data
    INT32*  size,    // IN/OUT: remaining bytes in the input buffer
    int     count    // IN: number of bytes to get
)
{
    if((*size -= count) >= 0)
    {
        memcpy(target, *buffer, count);
```

```
        *buffer += count;
        return TPM_RC_SUCCESS;
    }
    return TPM_RC_INSUFFICIENT;
}

//*** MarshalBytes()
// Marshal an array of bytes.
static UINT16 MarshalBytes(UINT8* source, UINT8** buffer, INT32* size, int32_t count)
{
    if(buffer != NULL)
    {
        if(size != NULL && (size -= count) < 0)
            return 0;
        memcpy(*buffer, source, count);
        *buffer += count;
    }
    return (UINT16)count;
}

//*** ArrayUnmarshal()
// Unmarshal an array. The 'index' is of the form: 'type'_ARRAY_MARSHAL_INDEX.
static TPM_RC ArrayUnmarshal(UINT16  index,   // IN: the type of the array
                             UINT8*  target,  // IN: target for the data
                             UINT8** buffer,  // IN/OUT: place to get the data
                             INT32*  size,    // IN/OUT: remaining unmarshal data
                             UINT32  count    // IN: number of values of 'index' to
                                              //      unmarshal
)
{
    marshalIndex_t which  = ArrayLookupTable[index & NULL_MASK].type;
    UINT16         stride = ArrayLookupTable[index & NULL_MASK].stride;
    TPM_RC         result;
    //
    if(stride == 1)  // A byte array
        result = UnmarshalBytes(target, buffer, size, count);
    else
    {
        which |= index & NULL_FLAG;
        for(result = TPM_RC_SUCCESS; count > 0; target += stride, count--)
            if(!IS_SUCCESS(Unmarshal(which, target, buffer, size)))
                break;
    }
    return result;
}

//*** ArrayMarshal()
static UINT16 ArrayMarshal(UINT16  index,   // IN: the type of the array
                           UINT8*  source,  // IN: source of the data
                           UINT8** buffer,  // IN/OUT: place to put the data
                           INT32*  size,    // IN/OUT: amount of space for the data
                           UINT32 count  // IN: number of values of 'index' to marshal
)
{
    marshalIndex_t which  = ArrayLookupTable[index & NULL_MASK].type;
    UINT16         stride = ArrayLookupTable[index & NULL_MASK].stride;
    UINT16         retVal;
    //
    if(stride == 1)  // A byte array
        return MarshalBytes(source, buffer, size, count);
    which |= index & NULL_FLAG;
    for(retVal = 0; count > 0; source += stride, count--)
        retVal += Marshal(which, source, buffer, size);

    return retVal;
}
```

```
//***UnmarshalUnion()
TPM_RC
UnmarshalUnion(UINT16  typeIndex,  // IN: the thing to unmarshal
               void*   target,     // IN: were the data goes to
               UINT8** buffer,     // IN/OUT: the data source buffer
               INT32*  size,       // IN/OUT: the remaining size
               UINT32  selector)
{
    int               i;
    UnionMarshal_mst* ut = GetUnionDescriptor(typeIndex);
    marshalIndex_t    selected;
    //
    for(i = 0; i < ut->countOfselectors; i++)
    {
        if(selector == ut->selectors[i])
        {
            UINT8* offset = ((UINT8*)ut) + ut->offsetOfUnmarshalTypes;
            // Get the selected thing to unmarshal
            selected = ((marshalIndex_t*)offset)[i];
            if(ut->modifiers & IS_ARRAY_UNION)
                return UnmarshalBytes(target, buffer, size, selected);
            else
            {
                // Propagate NULL_FLAG if the null flag was
                // propagated to the structure containing the union
                selected |= (typeIndex & NULL_FLAG);
                return Unmarshal(selected, target, buffer, size);
            }
        }
    }
    // Didn't find the value.
    return TPM_RC_SELECTOR;
}

//*** MarshalUnion()
UINT16
MarshalUnion(UINT16  typeIndex,  // IN: the thing to marshal
             void*   source,     // IN: were the data comes from
             UINT8** buffer,     // IN/OUT: the data source buffer
             INT32*  size,       // IN/OUT: the remaining size
             UINT32  selector    // IN: the union selector
)
{
    int               i;
    UnionMarshal_mst* ut = GetUnionDescriptor(typeIndex);
    marshalIndex_t    selected;
    //
    for(i = 0; i < ut->countOfselectors; i++)
    {
        if(selector == ut->selectors[i])
        {
            UINT8* offset = ((UINT8*)ut) + ut->offsetOfUnmarshalTypes;
            // Get the selected thing to unmarshal
            selected = ((marshalIndex_t*)offset)[i];
            if(ut->modifiers & IS_ARRAY_UNION)
                return MarshalBytes(source, buffer, size, selected);
            else
                return Marshal(selected, source, buffer, size);
        }
    }
    if(size != NULL)
        *size = -1;
    return 0;
}
```

```
TPM_RC
UnmarshalInteger(int     iSize,   // IN: Number of bytes in the integer
                 void*   target,  // OUT: receives the integer
                 UINT8** buffer,  // IN/OUT: source of the data
                 INT32*  size,    // IN/OUT: amount of data available
                 UINT32* value    // OUT: optional copy of 'target'
)
{
    // This is just to save typing
#  define _MB_ (*buffer)
    // The size is a power of two so convert to regular integer
    int bytes = (1 << (iSize & SIZE_MASK));
    //
    // Check to see if there is enough data to fulfill the request
    if((*size -= bytes) >= 0)
    {
        // The most comon size
        if(bytes == 4)
        {
            *((UINT32*)target) =
                (UINT32)((((((_MB_[0] << 8) | _MB_[1]) << 8) | _MB_[2]) << 8)
                         | _MB_[3]);
            // If a copy is needed, copy it.
            if(value != NULL)
                *value = *((UINT32*)target);
        }
        else if(bytes == 2)
        {
            *((UINT16*)target) = (UINT16)((_MB_[0] << 8) | _MB_[1]);
            // If a copy is needed, copy with the appropriate sign extension
            if(value != NULL)
            {
                if(iSize & IS_SIGNED)
                    *value = (UINT32)(*((INT16*)target));
                else
                    *value = (UINT32)(*((UINT16*)target));
            }
        }
        else if(bytes == 1)
        {
            *((UINT8*)target) = (UINT8)_MB_[0];
            // If a copy is needed, copy with the appropriate sign extension
            if(value != NULL)
            {
                if(iSize & IS_SIGNED)
                    *value = (UINT32)(*((INT8*)target));
                else
                    *value = (UINT32)(*((UINT8*)target));
            }
        }
        else
        {
            // There is no input type that is a 64-bit value other than a UINT64. So
            // there is no reason to do anything other than unmarshal it.
            *((UINT64*)target) = BYTE_ARRAY_TO_UINT64(*buffer);
        }
        *buffer += bytes;
        return TPM_RC_SUCCESS;
#  undef _MB_
    }
    return TPM_RC_INSUFFICIENT;
}

//*** Unmarshal()
// This is the function that performs unmarshaling of different numbered types. Each
// TPM type has a number. The number is used to lookup the address of the data
```

```
// structure that describes how to unmarshal that data type.
//
TPM_RC
Unmarshal(UINT16  typeIndex,  // IN: the thing to marshal
          void*   target,     // IN: were the data goes from
          UINT8** buffer,     // IN/OUT: the data source buffer
          INT32*  size        // IN/OUT: the remaining size
)
{
    const MarshalHeader_mst* sel;
    TPM_RC                   result;
    //
    sel = GetDescriptor(typeIndex);
    switch(sel->marshalType)
    {
        case UINT_MTYPE:
        {
            // A simple signed or unsigned integer value.
            return UnmarshalInteger(sel->modifiers, target, buffer, size, NULL);
        }
        case VALUES_MTYPE:
        {
            // This is the general-purpose structure that can handle things like
            // TPMI_DH_PARENT that has multiple ranges, multiple singles and a
            // 'null' value. When things cover a large range with holes in the range
            // they can be turned into multiple ranges. There is no option for a bit
            // field.
            // The structure is:
            //  typedef const struct ValuesMarshal_mst
            //  {
            //      UINT8           marshalType;        // VALUES_MTYPE
            //      UINT8           modifiers;
            //      UINT8           errorCode;
            //      UINT8           ranges;
            //      UINT8           singles;
            //      UINT32          values[1];
            //  } ValuesMarshal_mst;
            // Unmarshal the base type
            UINT32 val;
            if(IS_SUCCESS(
                   UnmarshalInteger(sel->modifiers, target, buffer, size, &val)))
            {
                ValuesMarshal_mst* vmt   = ((ValuesMarshal_mst*)sel);
                const UINT32*      check = vmt->values;
                //
                // if the TAKES_NULL flag is set, then the first entry in the values
                // list is the NULL value. It is not included in the 'ranges' or
                // 'singles' count.
                if((vmt->modifiers & TAKES_NULL) && (val == *check++))
                {
                    if((typeIndex & NULL_FLAG) == 0)
                        result = (TPM_RC)(sel->errorCode);
                }
                // No NULL value or input is not the NULL value
                else
                {
                    int i;
                    //
                    // Check all the min-max ranges.
                    for(i = vmt->ranges - 1; i >= 0; check = &check[2], i--)
                        if((UINT32)(val - check[0]) <= check[1])
                            break;
                    // if the input is in a selected range, then i >= 0
                    if(i < 0)
                    {
                        // Not in any range, so check sigles
```

```
                for(i = vmt->singles - 1; i >= 0; i--)
                    if(val == check[i])
                        break;
                }
                // If input not in range and not in any single so return error
                if(i < 0)
                    result = (TPM_RC)(sel->errorCode);
            }
        }
        break;
    }
    case TABLE_MTYPE:
    {
        // This is a table with or without bit checking. The input is checked
        // against each value in the table. If the value is in the table, and
        // a bits table is present, then the bit field is checked to see if the
        // indicated value is implemented. For example, if there is a table of
        // allowed RSA key sizes and the 2nd entry matches, then the 2nd bit in
        // the bit field is checked to see if that allowed size is implemented
        // in this TPM.
        //  typedef const struct TableMarshal_mst
        //  {
        //      UINT8           marshalType;        // TABLE_MTYPE
        //      UINT8           modifiers;
        //      UINT8           errorCode;
        //      UINT8           singles;
        //      UINT32          values[singles + 1 if TAKES_NULL];
        //  } TableMarshal_mst;

        UINT32 val;
        //
        // Unmarshal the base type
        if(IS_SUCCESS(
               UnmarshalInteger(sel->modifiers, target, buffer, size, &val)))
        {
            TableMarshal_mst* tmt   = ((TableMarshal_mst*)sel);
            const UINT32*     check = tmt->values;
            //
            // If this type has a null value, then it is the first value in the
            // list of values. It does not count in the count of values
            if((tmt->modifiers & TAKES_NULL) && (val == *check++))
            {
                if((typeIndex & NULL_FLAG) == 0)
                    result = (TPM_RC)(sel->errorCode);
            }
            else
            {
                int i;
                //
                // Process the singles
                for(i = tmt->singles - 1; i >= 0; i--)
                {
                    // does the input value match the value in the table
                    if(val == check[i])
                    {
                        // If there is an associated bit table, make sure that
                        // the corresponding bit is SET
                        if((HAS_BITS & tmt->modifiers)
                           && (!IS_BIT_SET32(i, &(check[tmt->singles]))))
                            // if not SET, then this is a failure.
                            i = -1;
                        break;
                    }
                }
                // error if not found or bit not SET
                if(i < 0)
```

```
                result = (TPM_RC)(sel->errorCode);
            }
        }
        break;
    }
    case MIN_MAX_MTYPE:
    {
        // A MIN_MAX is a range. It can have a bit field and a NULL value that is
        // outside of the range. If the input value is in the min-max range then
        // it is valid unless there is an associated bit field. Otherwise, it
        // it is only valid if the corresponding value in the bit field is SET.
        // The min value is 'values[0]' or 'values[1]' if there is a NULL value.
        // The max value is the value after min. The max value is in the table as
        // max minus min. This allows 'val' to be subtracted from min and then
        // checked against max with one unsigned comparison. If present, the bit
        // field will be the first 'values' after max.
        //  typedef const struct MinMaxMarshal_mst
        //  {
        //      UINT8           marshalType;        // MIN_MAX_MTYPE
        //      UINT8           modifiers;
        //      UINT8           errorCode;
        //      UINT32          values[2 + 1 if TAKES_NULL];
        //  } MinMaxMarshal_mst;
        UINT32 val;
        //
        // A min-max has a range. It can have a bit-field that is indexed to the
        // min value (something that matches min has a bit at 0. This is useful
        // for algorithms. The min-max define a range of algorithms to be checked
        // and the bit field can check to see if the algorithm in that range is
        // allowed.
        if(IS_SUCCESS(
               UnmarshalInteger(sel->modifiers, target, buffer, size, &val)))
        {
            MinMaxMarshal_mst* mmt   = (MinMaxMarshal_mst*)sel;
            const UINT32*      check = mmt->values;
            //
            // If this type takes a NULL, see if it matches. This
            if((mmt->modifiers & TAKES_NULL) && (val == *check++))
            {
                if((typeIndex & NULL_FLAG) == 0)
                    result = (TPM_RC)(mmt->errorCode);
            }
            else
            {
                val -= *check;
                if((val > check[1])
                   || ((mmt->modifiers & HAS_BITS)
                       && !IS_BIT_SET32(val, &check[2])))
                    result = (TPM_RC)(mmt->errorCode);
            }
        }
        break;
    }
    case ATTRIBUTES_MTYPE:
    {
        //  This is used for TPMA values.
        UINT32                  mask;
        AttributesMarshal_mst* amt = (AttributesMarshal_mst*)sel;
        //
        if(IS_SUCCESS(
               UnmarshalInteger(sel->modifiers, target, buffer, size, &mask)))
        {
            if((mask & amt->attributeMask) != 0)
                result = TPM_RC_RESERVED_BITS;
        }
        break;
```

```
}
case STRUCTURE_MTYPE:
{
    // A structure (not a union). A structure has elements (one defined per
    // row). Three UINT16 values are used for each row. The first indicates
    // the type of the entry. They choices are: simple, union, or array. A
    // simple type can be a simple integer or another structure. It can also
    // be a specific "interface type." For example, when a structure entry is
    // a value that is used define the dimension of an array, the entry of
    // the structure will reference a "synthetic" interface type, most often
    // a min-max value. If the type of the entry is union or array, then the
    // first value indicates which of the previous elements provides the union
    // selector or the array dimension. That previous entry is referenced in
    // the unmarshaled structure in memory (Not the marshaled buffer). The
    // previous entry indicates the location in the structure of the value.
    // The second entry of each structure entry indicated the index of the
    // type associated with the entry. This is an index into the array of
    // arrays or the union table (merged with the normal table in this
    // implementation). The final entry is the offset in the unmarshaled
    // structure where the value is located. This is the offsetof(STRUcTURE,
    // element). This value is added to the input 'target' or 'source' value
    // to determine where the value goes.
    StructMarshal_mst* mst = (StructMarshal_mst*)sel;
    int                i;
    const UINT16*      value = mst->values;
    //
    for(result = TPM_RC_SUCCESS, i = mst->elements;
        (TPM_RC_SUCCESS == result) && (i > 0);
        value = &value[3], i--)
    {
        UINT16         descriptor = value[0];
        marshalIndex_t index      = value[1];
        // The offset of the object in the structure is in the last value in
        // the triplet. Add that value to the start of the structure
        UINT8* offset = ((UINT8*)target) + value[2];
        //
        if((ELEMENT_PROPAGATE & descriptor) && (typeIndex & NULL_FLAG))
            index |= NULL_FLAG;
        switch(GET_ELEMENT_TYPE(descriptor))
        {
            case SIMPLE_STYPE:
            {
                result = Unmarshal(index, offset, buffer, size);
                break;
            }
            case UNION_STYPE:
            {
                UINT32 choice;
                //
                // Get the selector or array dimension value
                choice = GetSelector(target, mst->values, descriptor);
                result = UnmarshalUnion(index, offset, buffer, size, choice);
                break;
            }
            case ARRAY_STYPE:
            {
                UINT32 dimension;
                //
                dimension = GetSelector(target, mst->values, descriptor);
                result =
                    ArrayUnmarshal(index, offset, buffer, size, dimension);
                break;
            }
            default:
                result = TPM_RC_FAILURE;
                break;
```

```
            }
        }
        break;
    }
    case TPM2B_MTYPE:
    {
        // A primitive TPM2B. A size and byte buffer. The single value (other than
        // the tag) references the synthetic 'interface' value for the size
        // parameter.
        Tpm2bMarshal_mst* m2bt = (Tpm2bMarshal_mst*)sel;
        //
        if(IS_SUCCESS(Unmarshal(m2bt->sizeIndex, target, buffer, size)))
            result = UnmarshalBytes(
                ((TPM2B*)target)->buffer, buffer, size, *((UINT16*)target));
        break;
    }
    case TPM2BS_MTYPE:
    {
        // This is used when a TPM2B contains a structure.
        Tpm2bsMarshal_mst* m2bst = (Tpm2bsMarshal_mst*)sel;
        INT32              count;
        //
        if(IS_SUCCESS(Unmarshal(m2bst->sizeIndex, target, buffer, size)))
        {
            // fetch the size value and convert it to a 32-bit count value
            count = (int32_t) * ((UINT16*)target);
            if(count == 0)
            {
                if(m2bst->modifiers & SIZE_EQUAL)
                    result = TPM_RC_SIZE;
            }
            else if((*size -= count) >= 0)
            {
                marshalIndex_t index = m2bst->dataIndex;
                //
                // If this type propigates a null (PROPIGATE_NULL), propigate it
                if((m2bst->modifiers & PROPAGATE_NULL) && (typeIndex & typeIndex))
                    index |= NULL_FLAG;
                // The structure might not start two bytes after the start of the
                // size field. The offset to the start of the structure is between
                // 2 and 8 bytes. This is encoded into the low 4 bits of the
                // modifiers byte byte
                if(IS_SUCCESS(Unmarshal(
                       index,
                       ((UINT8*)target) + (m2bst->modifiers & OFFSET_MASK),
                       buffer,
                       &count)))
                {
                    if(count != 0)
                        result = TPM_RC_SIZE;
                }
            }
            else
                result = TPM_RC_INSUFFICIENT;
        }
        break;
    }
    case LIST_MTYPE:
    {
        // Used for a list. A list is a qualified 32-bit 'count' value followed
        // by a type indicator.
        ListMarshal_mst* mlt   = (ListMarshal_mst*)sel;
        marshalIndex_t   index = mlt->arrayRef;
        //
        if(IS_SUCCESS(Unmarshal(mlt->sizeIndex, target, buffer, size)))
        {
```

```
                // If this type propigates a null (PROPIGATE_NULL), propigate it
                if((mlt->modifiers & PROPAGATE_NULL) && (typeIndex & NULL_FLAG))
                    index |= NULL_FLAG;
                result =
                    ArrayUnmarshal(index,
                                   ((UINT8*)target) + (mlt->modifiers & OFFSET_MASK),
                                   buffer,
                                   size,
                                   *((UINT32*)target));
            }
            break;
        }
        case NULL_MTYPE:
        {
            result = TPM_RC_SUCCESS;
            break;
        }
#  if 0
        case COMPOSITE_MTYPE:
        {
            CompositeMarshal_mst    *mct = (CompositeMarshal_mst *)sel;
            int                      i;
            UINT8                   *buf = *buffer;
            INT32                    sz = *size;
        //
            result = TPM_RC_VALUE;
            for(i = GET_ELEMENT_COUNT(mct->modifiers) - 1; i <= 0; i--)
            {
                marshalIndex_t      index = mct->types[i];
            //
                // This type might take a null so set it in each called value, just
                // in case it is needed in that value. Only one value in each
                // composite should have the takes null SET.
                index |= typeIndex & NULL_MASK;
                result = Unmarshal(index, target, buffer, size);
                if(result == TPM_RC_SUCCESS)
                    break;
                // Each of the composite values does its own unmarshaling. This
                // has some execution overhead if it is unmarshaled multiple times
                // but it saves code size in not having to reproduce the various
                // unmarshaling types that can be in a composite. So, what this means
                // is that the buffer pointer and size have to be reset for each
                // unmarshaled value.
                *buffer = buf;
                *size = sz;
            }
            break;
        }
#  endif  // 0
        default:
        {
            result = TPM_RC_FAILURE;
            break;
        }
    }
    return result;
}

//*** Marshal()
// This is the function that drives marshaling of output. Because there is no
// validation of the output, there is a lot less code.
UINT16 Marshal(UINT16  typeIndex,  // IN: the thing to marshal
               void*   source,     // IN: were the data comes from
               UINT8** buffer,     // IN/OUT: the data source buffer
               INT32*  size        // IN/OUT: the remaining size
)
```

```
{
#  define _source ((UINT8*)source)

    const MarshalHeader_mst* sel;
    UINT16                   retVal;
    //
    sel = GetDescriptor(typeIndex);
    switch(sel->marshalType)
    {
        case VALUES_MTYPE:
        case UINT_MTYPE:
        case TABLE_MTYPE:
        case MIN_MAX_MTYPE:
        case ATTRIBUTES_MTYPE:
        case COMPOSITE_MTYPE:
        {
#  if BIG_ENDIAN_TPM
#    define MM16 0
#    define MM32 0
#    define MM64 0
#  else
            // These flip the constant index values so that they count in reverse
order when doing
            // little-endian stuff
#    define MM16 1
#    define MM32 3
#    define MM64 7
#  endif
// Just change the name and cast the type of the input parameters for typing purposes
#  define mb      (*buffer)
#  define _source ((UINT8*)source)
            retVal = (1 << (sel->modifiers & SIZE_MASK));
            if(buffer != NULL)
            {
                if((size == NULL) || ((*size -= retVal) >= 0))
                {
                    if(retVal == 4)
                    {
                        mb[0 ^ MM32] = _source[0];
                        mb[1 ^ MM32] = _source[1];
                        mb[2 ^ MM32] = _source[2];
                        mb[3 ^ MM32] = _source[3];
                    }
                    else if(retVal == 2)
                    {
                        mb[0 ^ MM16] = _source[0];
                        mb[1 ^ MM16] = _source[1];
                    }
                    else if(retVal == 1)
                        mb[0] = _source[0];
                    else
                    {
                        mb[0 ^ MM64] = _source[0];
                        mb[1 ^ MM64] = _source[1];
                        mb[2 ^ MM64] = _source[2];
                        mb[3 ^ MM64] = _source[3];
                        mb[4 ^ MM64] = _source[4];
                        mb[5 ^ MM64] = _source[5];
                        mb[6 ^ MM64] = _source[6];
                        mb[7 ^ MM64] = _source[7];
                    }
                    *buffer += retVal;
                }
            }
            break;
        }
```

```
case STRUCTURE_MTYPE:
{
    //#define _mst  ((StructMarshal_mst *)sel)
    StructMarshal_mst* mst = ((StructMarshal_mst*)sel);
    int                i;
    const UINT16*      value = mst->values;

    //
    for(retVal = 0, i = mst->elements; i > 0; value = &value[3], i--)
    {
        UINT16         des    = value[0];
        marshalIndex_t index  = value[1];
        UINT8*         offset = _source + value[2];
        //
        switch(GET_ELEMENT_TYPE(des))
        {
            case UNION_STYPE:
            {
                UINT32 choice;
                //
                choice = GetSelector(source, mst->values, des);
                retVal += MarshalUnion(index, offset, buffer, size, choice);
                break;
            }
            case ARRAY_STYPE:
            {
                UINT32 count;
                //
                count = GetSelector(source, mst->values, des);
                retVal += ArrayMarshal(index, offset, buffer, size, count);
                break;
            }
            case SIMPLE_STYPE:
            default:
            {
                // This is either another structure or a simple type
                retVal += Marshal(index, offset, buffer, size);
                break;
            }
        }
    }
    break;
}
case TPM2B_MTYPE:
{
    // Get the number of bytes being marshaled
    INT32 val = (int32_t) * ((UINT16*)source);
    //
    retVal = Marshal(UINT16_MARSHAL_REF, source, buffer, size);

    // This is a standard 2B with a byte buffer
    retVal += MarshalBytes(((TPM2B*)_source)->buffer, buffer, size, val);
    break;
}
case TPM2BS_MTYPE:  // A structure in a TPM2B
{
    Tpm2bsMarshal_mst* m2bst = (Tpm2bsMarshal_mst*)sel;
    UINT8*             offset;
    UINT16             amount;
    UINT8*             marshaledSize;
    //
    // Save the address of where the size should go
    marshaledSize = *buffer;

    // marshal the size (checks the space and advanced the pointer)
    retVal = Marshal(UINT16_MARSHAL_REF, source, buffer, size);
```

```
            // This gets the 'offsetof' the structure to marshal. It was placed in the
            // modifiers byte because the offset from the start of the TPM2B to the
            // start of the structure is going to be less than 8 and the modifiers
            // byte isn't needed for anything else.
            offset = _source + (m2bst->modifiers & SIGNED_MASK);

            // Marshal the structure and get its size
            amount = Marshal(m2bst->dataIndex, offset, buffer, size);

            // put the size in the space used when the size was marshaled.
            if(buffer != NULL)
                UINT16_TO_BYTE_ARRAY(amount, marshaledSize);
            retVal += amount;
            break;
        }
        case LIST_MTYPE:
        {
            ListMarshal_mst* mlt    = ((ListMarshal_mst*)sel);
            UINT8*           offset = _source + (mlt->modifiers & SIGNED_MASK);
            retVal = Marshal(UINT32_MARSHAL_REF, source, buffer, size);
            retVal += ArrayMarshal((marshalIndex_t)(mlt->arrayRef),
                                   offset,
                                   buffer,
                                   size,
                                   *((UINT32*)source));
            break;
        }
        case NULL_MTYPE:
            retVal = 0;
            break;
        case ERROR_MTYPE:
        default:
        {
            if(size != NULL)
                *size = -1;
            retVal = 0;
            break;
        }
    }
    return retVal;
}

#endif  // TABLE_DRIVEN_MARSHAL
```

7.197 /tpm/src/support/TableMarshalData.c

```
// This file contains the data initializer used for the table-driven marshaling code.

#include "Tpm.h"

#if TABLE_DRIVEN_MARSHAL
#  include "TableMarshal.h"
#  include "Marshal.h"

// The array marshaling table
ArrayMarshal_mst ArrayLookupTable[] = {ARRAY_MARSHAL_ENTRY(UINT8),
                                       ARRAY_MARSHAL_ENTRY(TPM_CC),
                                       ARRAY_MARSHAL_ENTRY(TPMA_CC),
                                       ARRAY_MARSHAL_ENTRY(TPM_ALG_ID),
                                       ARRAY_MARSHAL_ENTRY(TPM_HANDLE),
                                       ARRAY_MARSHAL_ENTRY(TPM2B_DIGEST),
                                       ARRAY_MARSHAL_ENTRY(TPMT_HA),
                                       ARRAY_MARSHAL_ENTRY(TPMS_PCR_SELECTION),
                                       ARRAY_MARSHAL_ENTRY(TPMS_ALG_PROPERTY),
```

```
                                        ARRAY_MARSHAL_ENTRY(TPMS_TAGGED_PROPERTY),
                                        ARRAY_MARSHAL_ENTRY(TPMS_TAGGED_PCR_SELECT),
                                        ARRAY_MARSHAL_ENTRY(TPM_ECC_CURVE),
                                        ARRAY_MARSHAL_ENTRY(TPMS_TAGGED_POLICY),
                                        ARRAY_MARSHAL_ENTRY(TPMS_ACT_DATA),
                                        ARRAY_MARSHAL_ENTRY(TPMS_AC_OUTPUT)};

// The main marshaling structure
MarshalData_st MarshalData = {
    // UINT8_DATA
    {UINT_MTYPE, 0},
    // UINT16_DATA
    {UINT_MTYPE, 1},
    // UINT32_DATA
    {UINT_MTYPE, 2},
    // UINT64_DATA
    {UINT_MTYPE, 3},
    // INT8_DATA
    {UINT_MTYPE, 0 + IS_SIGNED},
    // INT16_DATA
    {UINT_MTYPE, 1 + IS_SIGNED},
    // INT32_DATA
    {UINT_MTYPE, 2 + IS_SIGNED},
    // INT64_DATA
    {UINT_MTYPE, 3 + IS_SIGNED},
    // UINT0_DATA
    {NULL_MTYPE, 0},
    // TPM_ECC_CURVE_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES | TAKES_NULL | HAS_BITS,
     (UINT8)TPM_RC_CURVE,
     {TPM_ECC_NONE,
      RANGE(1, 32, UINT16),
      (UINT32)((ECC_NIST_P192 << 0) | (ECC_NIST_P224 << 1) | (ECC_NIST_P256 << 2)
               | (ECC_NIST_P384 << 3) | (ECC_NIST_P521 << 4) | (ECC_BN_P256 << 15)
               | (ECC_BN_P638 << 16) | (ECC_SM2_P256 << 31))}},
    // TPM_CLOCK_ADJUST_DATA
    {MIN_MAX_MTYPE,
     ONE_BYTES | IS_SIGNED,
     (UINT8)TPM_RC_VALUE,
     {RANGE(TPM_CLOCK_COARSE_SLOWER, TPM_CLOCK_COARSE_FASTER, INT8)}},
    // TPM_EO_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_VALUE,
     {RANGE(TPM_EO_EQ, TPM_EO_BITCLEAR, UINT16)}},
    // TPM_SU_DATA
    {TABLE_MTYPE, TWO_BYTES, (UINT8)TPM_RC_VALUE, 2, {TPM_SU_CLEAR, TPM_SU_STATE}},
    // TPM_SE_DATA
    {TABLE_MTYPE,
     ONE_BYTES,
     (UINT8)TPM_RC_VALUE,
     3,
     {TPM_SE_HMAC, TPM_SE_POLICY, TPM_SE_TRIAL}},
    // TPM_CAP_DATA
    {VALUES_MTYPE,
     FOUR_BYTES,
     (UINT8)TPM_RC_VALUE,
     1,
     1,
     {RANGE(TPM_CAP_ALGS, TPM_CAP_ACT, UINT32), TPM_CAP_VENDOR_PROPERTY}},
    // TPMA_ALGORITHM_DATA
    {ATTRIBUTES_MTYPE, FOUR_BYTES, 0xFFFFF8F0},
    // TPMA_OBJECT_DATA
    {ATTRIBUTES_MTYPE, FOUR_BYTES, 0xFFF0F001},
    // TPMA_SESSION_DATA
```

```
{ATTRIBUTES_MTYPE, ONE_BYTES, 0x00000018},
// TPMA_ACT_DATA
{ATTRIBUTES_MTYPE, FOUR_BYTES, 0xFFFFFFFC},
// TPMI_YES_NO_DATA
{TABLE_MTYPE, ONE_BYTES, (UINT8)TPM_RC_VALUE, 2, {NO, YES}},
// TPMI_DH_OBJECT_DATA
{VALUES_MTYPE,
 FOUR_BYTES | TAKES_NULL,
 (UINT8)TPM_RC_VALUE,
 2,
 0,
 {TPM_RH_NULL,
  RANGE(TRANSIENT_FIRST, TRANSIENT_LAST, UINT32),
  RANGE(PERSISTENT_FIRST, PERSISTENT_LAST, UINT32)}},
// TPMI_DH_PARENT_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 6,
 8,
 {RANGE(TRANSIENT_FIRST, TRANSIENT_LAST, UINT32),
  RANGE(PERSISTENT_FIRST, PERSISTENT_LAST, UINT32),
  RANGE(SVN_OWNER_FIRST, SVN_OWNER_LAST, UINT32),
  RANGE(SVN_ENDORSEMENT_FIRST, SVN_ENDORSEMENT_LAST, UINT32),
  RANGE(SVN_PLATFORM_FIRST, SVN_PLATFORM_LAST, UINT32),
  RANGE(SVN_NULL_FIRST, SVN_NULL_LAST, UINT32),
  TPM_RH_OWNER,
  TPM_RH_ENDORSEMENT,
  TPM_RH_PLATFORM,
  TPM_RH_NULL,
  TPM_RH_FW_OWNER,
  TPM_RH_FW_ENDORSEMENT,
  TPM_RH_FW_PLATFORM,
  TPM_RH_FW_NULL}},
// TPMI_DH_PERSISTENT_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(PERSISTENT_FIRST, PERSISTENT_LAST, UINT32)}},
// TPMI_DH_ENTITY_DATA
{VALUES_MTYPE,
 FOUR_BYTES | TAKES_NULL,
 (UINT8)TPM_RC_VALUE,
 5,
 4,
 {TPM_RH_NULL,
  RANGE(TRANSIENT_FIRST, TRANSIENT_LAST, UINT32),
  RANGE(PERSISTENT_FIRST, PERSISTENT_LAST, UINT32),
  RANGE(NV_INDEX_FIRST, NV_INDEX_LAST, UINT32),
  RANGE(PCR_FIRST, PCR_LAST, UINT32),
  RANGE(TPM_RH_AUTH_00, TPM_RH_AUTH_FF, UINT32),
  TPM_RH_OWNER,
  TPM_RH_LOCKOUT,
  TPM_RH_ENDORSEMENT,
  TPM_RH_PLATFORM}},
// TPMI_DH_PCR_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES | TAKES_NULL,
 (UINT8)TPM_RC_VALUE,
 {TPM_RH_NULL, RANGE(PCR_FIRST, PCR_LAST, UINT32)}},
// TPMI_SH_AUTH_SESSION_DATA
{VALUES_MTYPE,
 FOUR_BYTES | TAKES_NULL,
 (UINT8)TPM_RC_VALUE,
 2,
 0,
```

```
 {TPM_RS_PW,
  RANGE(HMAC_SESSION_FIRST, HMAC_SESSION_LAST, UINT32),
  RANGE(POLICY_SESSION_FIRST, POLICY_SESSION_LAST, UINT32)}},
// TPMI_SH_HMAC_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(HMAC_SESSION_FIRST, HMAC_SESSION_LAST, UINT32)}},
// TPMI_SH_POLICY_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(POLICY_SESSION_FIRST, POLICY_SESSION_LAST, UINT32)}},
// TPMI_DH_CONTEXT_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 3,
 0,
 {RANGE(HMAC_SESSION_FIRST, HMAC_SESSION_LAST, UINT32),
  RANGE(POLICY_SESSION_FIRST, POLICY_SESSION_LAST, UINT32),
  RANGE(TRANSIENT_FIRST, TRANSIENT_LAST, UINT32)}},
// TPMI_DH_SAVED_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 2,
 3,
 {RANGE(HMAC_SESSION_FIRST, HMAC_SESSION_LAST, UINT32),
  RANGE(POLICY_SESSION_FIRST, POLICY_SESSION_LAST, UINT32),
  0x80000000,
  0x80000001,
  0x80000002}},
// TPMI_RH_HIERARCHY_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 4,
 8,
 {RANGE(SVN_OWNER_FIRST, SVN_OWNER_LAST, UINT32),
  RANGE(SVN_ENDORSEMENT_FIRST, SVN_ENDORSEMENT_LAST, UINT32),
  RANGE(SVN_PLATFORM_FIRST, SVN_PLATFORM_LAST, UINT32),
  RANGE(SVN_NULL_FIRST, SVN_NULL_LAST, UINT32),
  TPM_RH_OWNER,
  TPM_RH_ENDORSEMENT,
  TPM_RH_PLATFORM,
  TPM_RH_NULL,
  TPM_RH_FW_OWNER,
  TPM_RH_FW_ENDORSEMENT,
  TPM_RH_FW_PLATFORM,
  TPM_RH_FW_NULL}},
// TPMI_RH_ENABLES_DATA
{TABLE_MTYPE,
 FOUR_BYTES | TAKES_NULL,
 (UINT8)TPM_RC_VALUE,
 4,
 {TPM_RH_NULL,
  TPM_RH_OWNER,
  TPM_RH_ENDORSEMENT,
  TPM_RH_PLATFORM,
  TPM_RH_PLATFORM_NV}},
// TPMI_RH_HIERARCHY_AUTH_DATA
{TABLE_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 4,
```

```
 {TPM_RH_OWNER, TPM_RH_LOCKOUT, TPM_RH_ENDORSEMENT, TPM_RH_PLATFORM}},
// TPMI_RH_HIERARCHY_POLICY_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 1,
 4,
 {RANGE(TPM_RH_ACT_0, TPM_RH_ACT_F, UINT32),
  TPM_RH_OWNER,
  TPM_RH_LOCKOUT,
  TPM_RH_ENDORSEMENT,
  TPM_RH_PLATFORM}},
// TPMI_RH_BASE_HIERARCHY_DATA
{TABLE_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 3,
 {TPM_RH_OWNER, TPM_RH_ENDORSEMENT, TPM_RH_PLATFORM}},
// TPMI_RH_PLATFORM_DATA
{TABLE_MTYPE, FOUR_BYTES, (UINT8)TPM_RC_VALUE, 1, {TPM_RH_PLATFORM}},
// TPMI_RH_OWNER_DATA
{TABLE_MTYPE,
 FOUR_BYTES | TAKES_NULL,
 (UINT8)TPM_RC_VALUE,
 1,
 {TPM_RH_NULL, TPM_RH_OWNER}},
// TPMI_RH_ENDORSEMENT_DATA
{TABLE_MTYPE,
 FOUR_BYTES | TAKES_NULL,
 (UINT8)TPM_RC_VALUE,
 1,
 {TPM_RH_NULL, TPM_RH_ENDORSEMENT}},
// TPMI_RH_PROVISION_DATA
{TABLE_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 2,
 {TPM_RH_OWNER, TPM_RH_PLATFORM}},
// TPMI_RH_CLEAR_DATA
{TABLE_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 2,
 {TPM_RH_LOCKOUT, TPM_RH_PLATFORM}},
// TPMI_RH_NV_AUTH_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 1,
 2,
 {RANGE(NV_INDEX_FIRST, NV_INDEX_LAST, UINT32), TPM_RH_OWNER, TPM_RH_PLATFORM}},
// TPMI_RH_LOCKOUT_DATA
{TABLE_MTYPE, FOUR_BYTES, (UINT8)TPM_RC_VALUE, 1, {TPM_RH_LOCKOUT}},
// TPMI_RH_NV_INDEX_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 3,
 0,
 {RANGE(NV_INDEX_FIRST, NV_INDEX_LAST, UINT32),
  RANGE(EXTERNAL_NV_FIRST, EXTERNAL_NV_LAST, UINT32),
  RANGE(PERMANENT_NV_FIRST, PERMANENT_NV_LAST, UINT32)}},
// TPMI_RH_DEFINED_NV_INDEX_DATA
{VALUES_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
```

```
 2,
 0,
 {RANGE(NV_INDEX_FIRST, NV_INDEX_LAST, UINT32),
  RANGE(EXTERNAL_NV_FIRST, EXTERNAL_NV_LAST, UINT32)}},
// TPMI_RH_LEGACY_NV_INDEX_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(NV_INDEX_FIRST, NV_INDEX_LAST, UINT32)}},
// TPMI_RH_EXP_NV_INDEX_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(EXTERNAL_NV_FIRST, EXTERNAL_NV_LAST, UINT32)}},
// TPMI_RH_AC_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(AC_FIRST, AC_LAST, UINT32)}},
// TPMI_RH_ACT_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(TPM_RH_ACT_0, TPM_RH_ACT_F, UINT32)}},
// TPMI_ALG_HASH_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_HASH,
 {TPM_ALG_NULL,
  RANGE(4, 41, UINT16),
  (UINT32)((ALG_SHA1 << 0) | (ALG_SHA256 << 7) | (ALG_SHA384 << 8)
           | (ALG_SHA512 << 9) | (ALG_SM3_256 << 14)),
  (UINT32)((ALG_SHA3_256 << 3) | (ALG_SHA3_384 << 4) | (ALG_SHA3_512 << 5))}},
// TPMI_ALG_ASYM_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_ASYMMETRIC,
 {TPM_ALG_NULL,
  RANGE(1, 35, UINT16),
  (UINT32)((ALG_RSA << 0)),
  (UINT32)((ALG_ECC << 2))}},
// TPMI_ALG_SYM_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_SYMMETRIC,
 {TPM_ALG_NULL,
  RANGE(3, 38, UINT16),
  (UINT32)((ALG_AES << 3) | (ALG_XOR << 7) | (ALG_SM4 << 16)),
  (UINT32)((ALG_CAMELLIA << 3))}},
// TPMI_ALG_SYM_OBJECT_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_SYMMETRIC,
 {TPM_ALG_NULL,
  RANGE(3, 38, UINT16),
  (UINT32)((ALG_AES << 3) | (ALG_SM4 << 16)),
  (UINT32)((ALG_CAMELLIA << 3))}},
// TPMI_ALG_SYM_MODE_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_MODE,
 {TPM_ALG_NULL,
  RANGE(63, 68, UINT16),
  (UINT32)((ALG_CMAC << 0) | (ALG_CTR << 1) | (ALG_OFB << 2) | (ALG_CBC << 3)
           | (ALG_CFB << 4) | (ALG_ECB << 5))}},
// TPMI_ALG_KDF_DATA
```

```
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_KDF,
 {TPM_ALG_NULL,
  RANGE(7, 34, UINT16),
  (UINT32)((ALG_MGF1 << 0) | (ALG_KDF1_SP800_56A << 25) | (ALG_KDF2 << 26)
           | (ALG_KDF1_SP800_108 << 27))}},
// TPMI_ALG_SIG_SCHEME_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_SCHEME,
 {TPM_ALG_NULL,
  RANGE(5, 28, UINT16),
  (UINT32)((ALG_HMAC << 0) | (ALG_RSASSA << 15) | (ALG_RSAPSS << 17)
           | (ALG_ECDSA << 19) | (ALG_ECDAA << 21) | (ALG_SM2 << 22)
           | (ALG_ECSCHNORR << 23))}},
// TPMI_ECC_KEY_EXCHANGE_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_SCHEME,
 {TPM_ALG_NULL,
  RANGE(25, 29, UINT16),
  (UINT32)((ALG_ECDH << 0) | (ALG_SM2 << 2) | (ALG_ECMQV << 4))}},
// TPMI_ST_COMMAND_TAG_DATA
{TABLE_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_BAD_TAG,
 2,
 {TPM_ST_NO_SESSIONS, TPM_ST_SESSIONS}},
// TPMI_ALG_MAC_SCHEME_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_SYMMETRIC,
 {TPM_ALG_NULL,
  RANGE(4, 63, UINT16),
  (UINT32)((ALG_SHA1 << 0) | (ALG_SHA256 << 7) | (ALG_SHA384 << 8)
           | (ALG_SHA512 << 9) | (ALG_SM3_256 << 14)),
  (UINT32)((ALG_SHA3_256 << 3) | (ALG_SHA3_384 << 4) | (ALG_SHA3_512 << 5)
           | (ALG_CMAC << 27))}},
// TPMI_ALG_CIPHER_MODE_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_MODE,
 {TPM_ALG_NULL,
  RANGE(64, 68, UINT16),
  (UINT32)((ALG_CTR << 0) | (ALG_OFB << 1) | (ALG_CBC << 2) | (ALG_CFB << 3)
           | (ALG_ECB << 4))}},
// TPMS_EMPTY_DATA
{STRUCTURE_MTYPE, 1, {SET_ELEMENT_TYPE(SIMPLE_STYPE), UINT0_MARSHAL_REF, 0}},
// TPMS_ALGORITHM_DESCRIPTION_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPM_ALG_ID_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ALGORITHM_DESCRIPTION, alg)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMA_ALGORITHM_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ALGORITHM_DESCRIPTION, attributes))}},
// TPMU_HA_DATA
{9,
 IS_ARRAY_UNION,
 (UINT16)(offsetof(TPMU_HA_mst, marshalingTypes)),
 {(UINT32)TPM_ALG_SHA1,
  (UINT32)TPM_ALG_SHA256,
  (UINT32)TPM_ALG_SHA384,
  (UINT32)TPM_ALG_SHA512,
```

```
    (UINT32)TPM_ALG_SM3_256,
    (UINT32)TPM_ALG_SHA3_256,
    (UINT32)TPM_ALG_SHA3_384,
    (UINT32)TPM_ALG_SHA3_512,
    (UINT32)TPM_ALG_NULL},
   {(UINT16)(SHA1_DIGEST_SIZE),
    (UINT16)(SHA256_DIGEST_SIZE),
    (UINT16)(SHA384_DIGEST_SIZE),
    (UINT16)(SHA512_DIGEST_SIZE),
    (UINT16)(SM3_256_DIGEST_SIZE),
    (UINT16)(SHA3_256_DIGEST_SIZE),
    (UINT16)(SHA3_384_DIGEST_SIZE),
    (UINT16)(SHA3_512_DIGEST_SIZE),
    (UINT16)(0)}},
  // TPMT_HA_DATA
  {STRUCTURE_MTYPE,
   2,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
        | ELEMENT_PROPAGATE,
    TPMI_ALG_HASH_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_HA, hashAlg)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
    TPMU_HA_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_HA, digest))}},
  // TPM2B_DIGEST_DATA
  {TPM2B_MTYPE, Type00_MARSHAL_REF},
  // TPM2B_DATA_DATA
  {TPM2B_MTYPE, Type01_MARSHAL_REF},
  // TPM2B_EVENT_DATA
  {TPM2B_MTYPE, Type02_MARSHAL_REF},
  // TPM2B_MAX_BUFFER_DATA
  {TPM2B_MTYPE, Type03_MARSHAL_REF},
  // TPM2B_MAX_NV_BUFFER_DATA
  {TPM2B_MTYPE, Type04_MARSHAL_REF},
  // TPM2B_TIMEOUT_DATA
  {TPM2B_MTYPE, Type05_MARSHAL_REF},
  // TPM2B_IV_DATA
  {TPM2B_MTYPE, Type06_MARSHAL_REF},
  // NULL_UNION_DATA
  {0},
  // TPM2B_NAME_DATA
  {TPM2B_MTYPE, Type07_MARSHAL_REF},
  // TPMS_PCR_SELECT_DATA
  {STRUCTURE_MTYPE,
   2,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
    Type08_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_PCR_SELECT, sizeofSelect)),
    SET_ELEMENT_TYPE(ARRAY_STYPE) | SET_ELEMENT_NUMBER(0),
    UINT8_ARRAY_MARSHAL_INDEX,
    (UINT16)(offsetof(TPMS_PCR_SELECT, pcrSelect))}},
  // TPMS_PCR_SELECTION_DATA
  {STRUCTURE_MTYPE,
   3,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ALG_HASH_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_PCR_SELECTION, hash)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
    Type08_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_PCR_SELECTION, sizeofSelect)),
    SET_ELEMENT_TYPE(ARRAY_STYPE) | SET_ELEMENT_NUMBER(1),
    UINT8_ARRAY_MARSHAL_INDEX,
    (UINT16)(offsetof(TPMS_PCR_SELECTION, pcrSelect))}},
  // TPMT_TK_CREATION_DATA
  {STRUCTURE_MTYPE,
   3,
```

```
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  Type10_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_CREATION, tag)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMI_RH_HIERARCHY_MARSHAL_REF | NULL_FLAG,
  (UINT16)(offsetof(TPMT_TK_CREATION, hierarchy)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_CREATION, digest))}},
// TPMT_TK_VERIFIED_DATA
{STRUCTURE_MTYPE,
 3,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  Type11_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_VERIFIED, tag)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMI_RH_HIERARCHY_MARSHAL_REF | NULL_FLAG,
  (UINT16)(offsetof(TPMT_TK_VERIFIED, hierarchy)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_VERIFIED, digest))}},
// TPMT_TK_AUTH_DATA
{STRUCTURE_MTYPE,
 3,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  Type12_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_AUTH, tag)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMI_RH_HIERARCHY_MARSHAL_REF | NULL_FLAG,
  (UINT16)(offsetof(TPMT_TK_AUTH, hierarchy)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_AUTH, digest))}},
// TPMT_TK_HASHCHECK_DATA
{STRUCTURE_MTYPE,
 3,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  Type13_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_HASHCHECK, tag)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMI_RH_HIERARCHY_MARSHAL_REF | NULL_FLAG,
  (UINT16)(offsetof(TPMT_TK_HASHCHECK, hierarchy)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_TK_HASHCHECK, digest))}},
// TPMS_ALG_PROPERTY_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPM_ALG_ID_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ALG_PROPERTY, alg)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMA_ALGORITHM_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ALG_PROPERTY, algProperties))}},
// TPMS_TAGGED_PROPERTY_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPM_PT_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TAGGED_PROPERTY, property)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  UINT32_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TAGGED_PROPERTY, value))}},
// TPMS_TAGGED_PCR_SELECT_DATA
{STRUCTURE_MTYPE,
 3,
```

```
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPM_PT_PCR_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TAGGED_PCR_SELECT, tag)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
  Type08_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TAGGED_PCR_SELECT, sizeofSelect)),
  SET_ELEMENT_TYPE(ARRAY_STYPE) | SET_ELEMENT_NUMBER(1),
  UINT8_ARRAY_MARSHAL_INDEX,
  (UINT16)(offsetof(TPMS_TAGGED_PCR_SELECT, pcrSelect))}},
// TPMS_TAGGED_POLICY_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPM_HANDLE_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TAGGED_POLICY, handle)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPMT_HA_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TAGGED_POLICY, policyHash))}},
// TPMS_ACT_DATA_DATA
{STRUCTURE_MTYPE,
 3,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPM_HANDLE_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ACT_DATA, handle)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  UINT32_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ACT_DATA, timeout)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMA_ACT_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ACT_DATA, attributes))}},
// TPML_CC_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_CC, commandCodes)),
 Type15_MARSHAL_REF,
 TPM_CC_ARRAY_MARSHAL_INDEX},
// TPML_CCA_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_CCA, commandAttributes)),
 Type15_MARSHAL_REF,
 TPMA_CC_ARRAY_MARSHAL_INDEX},
// TPML_ALG_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_ALG, algorithms)),
 Type17_MARSHAL_REF,
 TPM_ALG_ID_ARRAY_MARSHAL_INDEX},
// TPML_HANDLE_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_HANDLE, handle)),
 Type18_MARSHAL_REF,
 TPM_HANDLE_ARRAY_MARSHAL_INDEX},
// TPML_DIGEST_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_DIGEST, digests)),
 Type19_MARSHAL_REF,
 TPM2B_DIGEST_ARRAY_MARSHAL_INDEX},
// TPML_DIGEST_VALUES_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_DIGEST_VALUES, digests)),
 Type20_MARSHAL_REF,
 TPMT_HA_ARRAY_MARSHAL_INDEX},
// TPML_PCR_SELECTION_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_PCR_SELECTION, pcrSelections)),
 Type20_MARSHAL_REF,
 TPMS_PCR_SELECTION_ARRAY_MARSHAL_INDEX},
// TPML_ALG_PROPERTY_DATA
```

```
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_ALG_PROPERTY, algProperties)),
 Type22_MARSHAL_REF,
 TPMS_ALG_PROPERTY_ARRAY_MARSHAL_INDEX},
// TPML_TAGGED_TPM_PROPERTY_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_TAGGED_TPM_PROPERTY, tpmProperty)),
 Type23_MARSHAL_REF,
 TPMS_TAGGED_PROPERTY_ARRAY_MARSHAL_INDEX},
// TPML_TAGGED_PCR_PROPERTY_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_TAGGED_PCR_PROPERTY, pcrProperty)),
 Type24_MARSHAL_REF,
 TPMS_TAGGED_PCR_SELECT_ARRAY_MARSHAL_INDEX},
// TPML_ECC_CURVE_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_ECC_CURVE, eccCurves)),
 Type25_MARSHAL_REF,
 TPM_ECC_CURVE_ARRAY_MARSHAL_INDEX},
// TPML_TAGGED_POLICY_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_TAGGED_POLICY, policies)),
 Type26_MARSHAL_REF,
 TPMS_TAGGED_POLICY_ARRAY_MARSHAL_INDEX},
// TPML_ACT_DATA_DATA
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_ACT_DATA, actData)),
 Type27_MARSHAL_REF,
 TPMS_ACT_DATA_ARRAY_MARSHAL_INDEX},
// TPMU_CAPABILITIES_DATA
{11,
 0,
 (UINT16)(offsetof(TPMU_CAPABILITIES_mst, marshalingTypes)),
 {(UINT32)TPM_CAP_ALGS,
  (UINT32)TPM_CAP_HANDLES,
  (UINT32)TPM_CAP_COMMANDS,
  (UINT32)TPM_CAP_PP_COMMANDS,
  (UINT32)TPM_CAP_AUDIT_COMMANDS,
  (UINT32)TPM_CAP_PCRS,
  (UINT32)TPM_CAP_TPM_PROPERTIES,
  (UINT32)TPM_CAP_PCR_PROPERTIES,
  (UINT32)TPM_CAP_ECC_CURVES,
  (UINT32)TPM_CAP_AUTH_POLICIES,
  (UINT32)TPM_CAP_ACT},
 {(UINT16)(TPML_ALG_PROPERTY_MARSHAL_REF),
  (UINT16)(TPML_HANDLE_MARSHAL_REF),
  (UINT16)(TPML_CCA_MARSHAL_REF),
  (UINT16)(TPML_CC_MARSHAL_REF),
  (UINT16)(TPML_CC_MARSHAL_REF),
  (UINT16)(TPML_PCR_SELECTION_MARSHAL_REF),
  (UINT16)(TPML_TAGGED_TPM_PROPERTY_MARSHAL_REF),
  (UINT16)(TPML_TAGGED_PCR_PROPERTY_MARSHAL_REF),
  (UINT16)(TPML_ECC_CURVE_MARSHAL_REF),
  (UINT16)(TPML_TAGGED_POLICY_MARSHAL_REF),
  (UINT16)(TPML_ACT_DATA_MARSHAL_REF)}},
// TPMS_CAPABILITY_DATA_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPM_CAP_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CAPABILITY_DATA, capability)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_CAPABILITIES_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CAPABILITY_DATA, data))}},
// TPMU_SET_CAPABILITIES_DATA
{0, 0, 0, {}, {}},
```

```
// TPMS_SET_CAPABILITY_DATA_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPM_CAP_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SET_CAPABILITY_DATA, setCapability)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_SET_CAPABILITIES_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SET_CAPABILITY_DATA, data))}},
// TPM2B_SET_CAPABILITY_DATA_DATA
{TPM2B_MTYPE, Type03_MARSHAL_REF},
// TPMS_CLOCK_INFO_DATA
{STRUCTURE_MTYPE,
 4,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(EIGHT_BYTES),
  UINT64_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CLOCK_INFO, clock)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  UINT32_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CLOCK_INFO, resetCount)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  UINT32_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CLOCK_INFO, restartCount)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
  TPMI_YES_NO_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CLOCK_INFO, safe))}},
// TPMS_TIME_INFO_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(EIGHT_BYTES),
  UINT64_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TIME_INFO, time)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPMS_CLOCK_INFO_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TIME_INFO, clockInfo))}},
// TPMS_TIME_ATTEST_INFO_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPMS_TIME_INFO_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TIME_ATTEST_INFO, time)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(EIGHT_BYTES),
  UINT64_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_TIME_ATTEST_INFO, firmwareVersion))}},
// TPMS_CERTIFY_INFO_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_NAME_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CERTIFY_INFO, name)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_NAME_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CERTIFY_INFO, qualifiedName))}},
// TPMS_QUOTE_INFO_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPML_PCR_SELECTION_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_QUOTE_INFO, pcrSelect)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_QUOTE_INFO, pcrDigest))}},
// TPMS_COMMAND_AUDIT_INFO_DATA
{STRUCTURE_MTYPE,
 4,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(EIGHT_BYTES),
```

```
  UINT64_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_COMMAND_AUDIT_INFO, auditCounter)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPM_ALG_ID_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_COMMAND_AUDIT_INFO, digestAlg)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_COMMAND_AUDIT_INFO, auditDigest)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_COMMAND_AUDIT_INFO, commandDigest))}},
// TPMS_SESSION_AUDIT_INFO_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
  TPMI_YES_NO_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SESSION_AUDIT_INFO, exclusiveSession)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SESSION_AUDIT_INFO, sessionDigest))}},
// TPMS_CREATION_INFO_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_NAME_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CREATION_INFO, objectName)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_CREATION_INFO, creationHash))}},
// TPMS_NV_CERTIFY_INFO_DATA
{STRUCTURE_MTYPE,
 3,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_NAME_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_CERTIFY_INFO, indexName)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  UINT16_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_CERTIFY_INFO, offset)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_MAX_NV_BUFFER_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_CERTIFY_INFO, nvContents))}},
// TPMS_NV_DIGEST_CERTIFY_INFO_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_NAME_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_DIGEST_CERTIFY_INFO, indexName)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_DIGEST_CERTIFY_INFO, nvDigest))}},
// TPMI_ST_ATTEST_DATA
{VALUES_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_VALUE,
 1,
 1,
 {RANGE(TPM_ST_ATTEST_NV, TPM_ST_ATTEST_CREATION, UINT16),
  TPM_ST_ATTEST_NV_DIGEST}},
// TPMU_ATTEST_DATA
{8,
 0,
 (UINT16)(offsetof(TPMU_ATTEST_mst, marshalingTypes)),
 {(UINT32)TPM_ST_ATTEST_CERTIFY,
  (UINT32)TPM_ST_ATTEST_CREATION,
  (UINT32)TPM_ST_ATTEST_QUOTE,
  (UINT32)TPM_ST_ATTEST_COMMAND_AUDIT,
```

```
    (UINT32)TPM_ST_ATTEST_SESSION_AUDIT,
    (UINT32)TPM_ST_ATTEST_TIME,
    (UINT32)TPM_ST_ATTEST_NV,
    (UINT32)TPM_ST_ATTEST_NV_DIGEST},
   {(UINT16)(TPMS_CERTIFY_INFO_MARSHAL_REF),
    (UINT16)(TPMS_CREATION_INFO_MARSHAL_REF),
    (UINT16)(TPMS_QUOTE_INFO_MARSHAL_REF),
    (UINT16)(TPMS_COMMAND_AUDIT_INFO_MARSHAL_REF),
    (UINT16)(TPMS_SESSION_AUDIT_INFO_MARSHAL_REF),
    (UINT16)(TPMS_TIME_ATTEST_INFO_MARSHAL_REF),
    (UINT16)(TPMS_NV_CERTIFY_INFO_MARSHAL_REF),
    (UINT16)(TPMS_NV_DIGEST_CERTIFY_INFO_MARSHAL_REF)}},
  // TPMS_ATTEST_DATA
  {STRUCTURE_MTYPE,
   7,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
    TPM_CONSTANTS32_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ATTEST, magic)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ST_ATTEST_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ATTEST, type)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_NAME_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ATTEST, qualifiedSigner)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_DATA_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ATTEST, extraData)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMS_CLOCK_INFO_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ATTEST, clockInfo)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(EIGHT_BYTES),
    UINT64_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ATTEST, firmwareVersion)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(1),
    TPMU_ATTEST_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ATTEST, attested))}},
  // TPM2B_ATTEST_DATA
  {TPM2B_MTYPE, Type28_MARSHAL_REF},
  // TPMS_AUTH_COMMAND_DATA
  {STRUCTURE_MTYPE,
   4,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
    TPMI_SH_AUTH_SESSION_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_AUTH_COMMAND, sessionHandle)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_NONCE_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_AUTH_COMMAND, nonce)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
    TPMA_SESSION_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_AUTH_COMMAND, sessionAttributes)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_AUTH_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_AUTH_COMMAND, hmac))}},
  // TPMS_AUTH_RESPONSE_DATA
  {STRUCTURE_MTYPE,
   3,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_NONCE_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_AUTH_RESPONSE, nonce)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
    TPMA_SESSION_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_AUTH_RESPONSE, sessionAttributes)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_AUTH_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_AUTH_RESPONSE, hmac))}},
  // TPMI_AES_KEY_BITS_DATA
```

```
{TABLE_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_VALUE,
 3,
 {192 * AES_192, 128 * AES_128, 256 * AES_256}},
// TPMI_SM4_KEY_BITS_DATA
{TABLE_MTYPE, TWO_BYTES, (UINT8)TPM_RC_VALUE, 1, {128 * SM4_128}},
// TPMI_CAMELLIA_KEY_BITS_DATA
{TABLE_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_VALUE,
 3,
 {192 * CAMELLIA_192, 128 * CAMELLIA_128, 256 * CAMELLIA_256}},
// TPMU_SYM_KEY_BITS_DATA
{6,
 0,
 (UINT16)(offsetof(TPMU_SYM_KEY_BITS_mst, marshalingTypes)),
 {(UINT32)TPM_ALG_AES,
  (UINT32)TPM_ALG_SM4,
  (UINT32)TPM_ALG_CAMELLIA,
  (UINT32)TPM_ALG_XOR,
  (UINT32)TPM_ALG_NULL},
 {(UINT16)(TPMI_AES_KEY_BITS_MARSHAL_REF),
  (UINT16)(TPMI_SM4_KEY_BITS_MARSHAL_REF),
  (UINT16)(TPMI_CAMELLIA_KEY_BITS_MARSHAL_REF),
  (UINT16)(TPMI_ALG_HASH_MARSHAL_REF),
  (UINT16)(UINT0_MARSHAL_REF)}},
// TPMU_SYM_MODE_DATA
{6,
 0,
 (UINT16)(offsetof(TPMU_SYM_MODE_mst, marshalingTypes)),
 {(UINT32)TPM_ALG_AES,
  (UINT32)TPM_ALG_SM4,
  (UINT32)TPM_ALG_CAMELLIA,
  (UINT32)TPM_ALG_XOR,
  (UINT32)TPM_ALG_NULL},
 {(UINT16)(TPMI_ALG_SYM_MODE_MARSHAL_REF | NULL_FLAG),
  (UINT16)(TPMI_ALG_SYM_MODE_MARSHAL_REF | NULL_FLAG),
  (UINT16)(TPMI_ALG_SYM_MODE_MARSHAL_REF | NULL_FLAG),
  (UINT16)(UINT0_MARSHAL_REF),
  (UINT16)(UINT0_MARSHAL_REF)}},
// TPMT_SYM_DEF_DATA
{STRUCTURE_MTYPE,
 3,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
      | ELEMENT_PROPAGATE,
  TPMI_ALG_SYM_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SYM_DEF, algorithm)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_SYM_KEY_BITS_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SYM_DEF, keyBits)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_SYM_MODE_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SYM_DEF, mode))}},
// TPMT_SYM_DEF_OBJECT_DATA
{STRUCTURE_MTYPE,
 3,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
      | ELEMENT_PROPAGATE,
  TPMI_ALG_SYM_OBJECT_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SYM_DEF_OBJECT, algorithm)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_SYM_KEY_BITS_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SYM_DEF_OBJECT, keyBits)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_SYM_MODE_MARSHAL_REF,
```

```
  (UINT16)(offsetof(TPMT_SYM_DEF_OBJECT, mode))}},
// TPM2B_SYM_KEY_DATA
{TPM2B_MTYPE, Type29_MARSHAL_REF},
// TPMS_SYMCIPHER_PARMS_DATA
{STRUCTURE_MTYPE,
 1,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPMT_SYM_DEF_OBJECT_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SYMCIPHER_PARMS, sym))}},
// TPM2B_LABEL_DATA
{TPM2B_MTYPE, Type30_MARSHAL_REF},
// TPMS_DERIVE_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_LABEL_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_DERIVE, label)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_LABEL_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_DERIVE, context))}},
// TPM2B_DERIVE_DATA
{TPM2B_MTYPE, Type31_MARSHAL_REF},
// TPM2B_SENSITIVE_DATA_DATA
{TPM2B_MTYPE, Type32_MARSHAL_REF},
// TPMS_SENSITIVE_CREATE_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_AUTH_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SENSITIVE_CREATE, userAuth)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_SENSITIVE_DATA_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SENSITIVE_CREATE, data))}},
// TPM2B_SENSITIVE_CREATE_DATA
{TPM2BS_MTYPE,
 (UINT8)(offsetof(TPM2B_SENSITIVE_CREATE, sensitive)) | SIZE_EQUAL,
 UINT16_MARSHAL_REF,
 TPMS_SENSITIVE_CREATE_MARSHAL_REF},
// TPMS_SCHEME_HASH_DATA
{STRUCTURE_MTYPE,
 1,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPMI_ALG_HASH_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SCHEME_HASH, hashAlg))}},
// TPMS_SCHEME_ECDAA_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPMI_ALG_HASH_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SCHEME_ECDAA, hashAlg)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  UINT16_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SCHEME_ECDAA, count))}},
// TPMI_ALG_KEYEDHASH_SCHEME_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_VALUE,
 {TPM_ALG_NULL,
  RANGE(5, 10, UINT16),
  (UINT32)((ALG_HMAC << 0) | (ALG_XOR << 5))}},
// TPMS_SCHEME_XOR_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPMI_ALG_HASH_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_SCHEME_XOR, hashAlg)),
```

```
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ALG_KDF_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_SCHEME_XOR, kdf))}},
  // TPMU_SCHEME_KEYEDHASH_DATA
  {3,
   0,
   (UINT16)(offsetof(TPMU_SCHEME_KEYEDHASH_mst, marshalingTypes)),
   {(UINT32)TPM_ALG_HMAC, (UINT32)TPM_ALG_XOR, (UINT32)TPM_ALG_NULL},
   {(UINT16)(TPMS_SCHEME_HMAC_MARSHAL_REF),
    (UINT16)(TPMS_SCHEME_XOR_MARSHAL_REF),
    (UINT16)(UINT0_MARSHAL_REF)}},
  // TPMT_KEYEDHASH_SCHEME_DATA
  {STRUCTURE_MTYPE,
   2,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
        | ELEMENT_PROPAGATE,
    TPMI_ALG_KEYEDHASH_SCHEME_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_KEYEDHASH_SCHEME, scheme)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
    TPMU_SCHEME_KEYEDHASH_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_KEYEDHASH_SCHEME, details))}},
  // TPMU_SIG_SCHEME_DATA
  {8,
   0,
   (UINT16)(offsetof(TPMU_SIG_SCHEME_mst, marshalingTypes)),
   {(UINT32)TPM_ALG_ECDAA,
    (UINT32)TPM_ALG_RSASSA,
    (UINT32)TPM_ALG_RSAPSS,
    (UINT32)TPM_ALG_ECDSA,
    (UINT32)TPM_ALG_SM2,
    (UINT32)TPM_ALG_ECSCHNORR,
    (UINT32)TPM_ALG_HMAC,
    (UINT32)TPM_ALG_NULL},
   {(UINT16)(TPMS_SIG_SCHEME_ECDAA_MARSHAL_REF),
    (UINT16)(TPMS_SIG_SCHEME_RSASSA_MARSHAL_REF),
    (UINT16)(TPMS_SIG_SCHEME_RSAPSS_MARSHAL_REF),
    (UINT16)(TPMS_SIG_SCHEME_ECDSA_MARSHAL_REF),
    (UINT16)(TPMS_SIG_SCHEME_SM2_MARSHAL_REF),
    (UINT16)(TPMS_SIG_SCHEME_ECSCHNORR_MARSHAL_REF),
    (UINT16)(TPMS_SCHEME_HMAC_MARSHAL_REF),
    (UINT16)(UINT0_MARSHAL_REF)}},
  // TPMT_SIG_SCHEME_DATA
  {STRUCTURE_MTYPE,
   2,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
        | ELEMENT_PROPAGATE,
    TPMI_ALG_SIG_SCHEME_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_SIG_SCHEME, scheme)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
    TPMU_SIG_SCHEME_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_SIG_SCHEME, details))}},
  // TPMU_KDF_SCHEME_DATA
  {5,
   0,
   (UINT16)(offsetof(TPMU_KDF_SCHEME_mst, marshalingTypes)),
   {(UINT32)TPM_ALG_MGF1,
    (UINT32)TPM_ALG_KDF1_SP800_56A,
    (UINT32)TPM_ALG_KDF2,
    (UINT32)TPM_ALG_KDF1_SP800_108,
    (UINT32)TPM_ALG_NULL},
   {(UINT16)(TPMS_KDF_SCHEME_MGF1_MARSHAL_REF),
    (UINT16)(TPMS_KDF_SCHEME_KDF1_SP800_56A_MARSHAL_REF),
    (UINT16)(TPMS_KDF_SCHEME_KDF2_MARSHAL_REF),
    (UINT16)(TPMS_KDF_SCHEME_KDF1_SP800_108_MARSHAL_REF),
    (UINT16)(UINT0_MARSHAL_REF)}},
  // TPMT_KDF_SCHEME_DATA
```

```
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
      | ELEMENT_PROPAGATE,
  TPMI_ALG_KDF_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_KDF_SCHEME, scheme)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_KDF_SCHEME_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_KDF_SCHEME, details))}},
// TPMI_ALG_ASYM_SCHEME_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_VALUE,
 {TPM_ALG_NULL,
  RANGE(20, 29, UINT16),
  (UINT32)((ALG_RSASSA << 0) | (ALG_RSAES << 1) | (ALG_RSAPSS << 2)
           | (ALG_OAEP << 3) | (ALG_ECDSA << 4) | (ALG_ECDH << 5)
           | (ALG_ECDAA << 6) | (ALG_SM2 << 7) | (ALG_ECSCHNORR << 8)
           | (ALG_ECMQV << 9))}},
// TPMU_ASYM_SCHEME_DATA
{11,
 0,
 (UINT16)(offsetof(TPMU_ASYM_SCHEME_mst, marshalingTypes)),
 {(UINT32)TPM_ALG_ECDH,
  (UINT32)TPM_ALG_ECMQV,
  (UINT32)TPM_ALG_ECDAA,
  (UINT32)TPM_ALG_RSASSA,
  (UINT32)TPM_ALG_RSAPSS,
  (UINT32)TPM_ALG_ECDSA,
  (UINT32)TPM_ALG_SM2,
  (UINT32)TPM_ALG_ECSCHNORR,
  (UINT32)TPM_ALG_RSAES,
  (UINT32)TPM_ALG_OAEP,
  (UINT32)TPM_ALG_NULL},
 {(UINT16)(TPMS_KEY_SCHEME_ECDH_MARSHAL_REF),
  (UINT16)(TPMS_KEY_SCHEME_ECMQV_MARSHAL_REF),
  (UINT16)(TPMS_SIG_SCHEME_ECDAA_MARSHAL_REF),
  (UINT16)(TPMS_SIG_SCHEME_RSASSA_MARSHAL_REF),
  (UINT16)(TPMS_SIG_SCHEME_RSAPSS_MARSHAL_REF),
  (UINT16)(TPMS_SIG_SCHEME_ECDSA_MARSHAL_REF),
  (UINT16)(TPMS_SIG_SCHEME_SM2_MARSHAL_REF),
  (UINT16)(TPMS_SIG_SCHEME_ECSCHNORR_MARSHAL_REF),
  (UINT16)(TPMS_ENC_SCHEME_RSAES_MARSHAL_REF),
  (UINT16)(TPMS_ENC_SCHEME_OAEP_MARSHAL_REF),
  (UINT16)(UINT0_MARSHAL_REF)}},
// TPMI_ALG_RSA_SCHEME_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_VALUE,
 {TPM_ALG_NULL,
  RANGE(20, 23, UINT16),
  (UINT32)((ALG_RSASSA << 0) | (ALG_RSAES << 1) | (ALG_RSAPSS << 2)
           | (ALG_OAEP << 3))}},
// TPMT_RSA_SCHEME_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
      | ELEMENT_PROPAGATE,
  TPMI_ALG_RSA_SCHEME_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_RSA_SCHEME, scheme)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_ASYM_SCHEME_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_RSA_SCHEME, details))}},
// TPMI_ALG_RSA_DECRYPT_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
```

```
 (UINT8)TPM_RC_VALUE,
 {TPM_ALG_NULL,
  RANGE(21, 23, UINT16),
  (UINT32)((ALG_RSAES << 0) | (ALG_OAEP << 2))}},
// TPMT_RSA_DECRYPT_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
      | ELEMENT_PROPAGATE,
  TPMI_ALG_RSA_DECRYPT_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_RSA_DECRYPT, scheme)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_ASYM_SCHEME_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_RSA_DECRYPT, details))}},
// TPM2B_PUBLIC_KEY_RSA_DATA
{TPM2B_MTYPE, Type33_MARSHAL_REF},
// TPMI_RSA_KEY_BITS_DATA
{TABLE_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_VALUE,
 3,
 {3072 * RSA_3072, 1024 * RSA_1024, 2048 * RSA_2048}},
// TPM2B_PRIVATE_KEY_RSA_DATA
{TPM2B_MTYPE, Type34_MARSHAL_REF},
// TPM2B_ECC_PARAMETER_DATA
{TPM2B_MTYPE, Type35_MARSHAL_REF},
// TPMS_ECC_POINT_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_ECC_PARAMETER_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ECC_POINT, x)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_ECC_PARAMETER_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_ECC_POINT, y))}},
// TPM2B_ECC_POINT_DATA
{TPM2BS_MTYPE,
 (UINT8)(offsetof(TPM2B_ECC_POINT, point)) | SIZE_EQUAL,
 UINT16_MARSHAL_REF,
 TPMS_ECC_POINT_MARSHAL_REF},
// TPMI_ALG_ECC_SCHEME_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_SCHEME,
 {TPM_ALG_NULL,
  RANGE(24, 29, UINT16),
  (UINT32)((ALG_ECDSA << 0) | (ALG_ECDH << 1) | (ALG_ECDAA << 2) | (ALG_SM2 << 3)
           | (ALG_ECSCHNORR << 4) | (ALG_ECMQV << 5))}},
// TPMI_ECC_CURVE_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES | TAKES_NULL | HAS_BITS,
 (UINT8)TPM_RC_CURVE,
 {TPM_ECC_NONE,
  RANGE(1, 32, UINT16),
  (UINT32)((ECC_NIST_P192 << 0) | (ECC_NIST_P224 << 1) | (ECC_NIST_P256 << 2)
           | (ECC_NIST_P384 << 3) | (ECC_NIST_P521 << 4) | (ECC_BN_P256 << 15)
           | (ECC_BN_P638 << 16) | (ECC_SM2_P256 << 31))}},
// TPMT_ECC_SCHEME_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
      | ELEMENT_PROPAGATE,
  TPMI_ALG_ECC_SCHEME_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_ECC_SCHEME, scheme)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_ASYM_SCHEME_MARSHAL_REF,
```

```
   (UINT16)(offsetof(TPMT_ECC_SCHEME, details))},
  // TPMS_ALGORITHM_DETAIL_ECC_DATA
  {STRUCTURE_MTYPE,
   11,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPM_ECC_CURVE_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, curveID)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    UINT16_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, keySize)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_KDF_SCHEME_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, kdf)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_ECC_SCHEME_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, sign)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, p)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, a)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, b)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, gX)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, gY)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, n)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ALGORITHM_DETAIL_ECC, h))}},
  // TPMS_SIGNATURE_RSA_DATA
  {STRUCTURE_MTYPE,
   2,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ALG_HASH_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_SIGNATURE_RSA, hash)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_PUBLIC_KEY_RSA_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_SIGNATURE_RSA, sig))}},
  // TPMS_SIGNATURE_ECC_DATA
  {STRUCTURE_MTYPE,
   3,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ALG_HASH_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_SIGNATURE_ECC, hash)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_SIGNATURE_ECC, signatureR)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_ECC_PARAMETER_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_SIGNATURE_ECC, signatureS))}},
  // TPMU_SIGNATURE_DATA
  {8,
   0,
   (UINT16)(offsetof(TPMU_SIGNATURE_mst, marshalingTypes)),
   {(UINT32)TPM_ALG_ECDAA,
    (UINT32)TPM_ALG_RSASSA,
    (UINT32)TPM_ALG_RSAPSS,
    (UINT32)TPM_ALG_ECDSA,
```

```
    (UINT32)TPM_ALG_SM2,
    (UINT32)TPM_ALG_ECSCHNORR,
    (UINT32)TPM_ALG_HMAC,
    (UINT32)TPM_ALG_NULL},
   {(UINT16)(TPMS_SIGNATURE_ECDAA_MARSHAL_REF),
    (UINT16)(TPMS_SIGNATURE_RSASSA_MARSHAL_REF),
    (UINT16)(TPMS_SIGNATURE_RSAPSS_MARSHAL_REF),
    (UINT16)(TPMS_SIGNATURE_ECDSA_MARSHAL_REF),
    (UINT16)(TPMS_SIGNATURE_SM2_MARSHAL_REF),
    (UINT16)(TPMS_SIGNATURE_ECSCHNORR_MARSHAL_REF),
    (UINT16)(TPMT_HA_MARSHAL_REF),
    (UINT16)(UINT0_MARSHAL_REF)}},
  // TPMT_SIGNATURE_DATA
  {STRUCTURE_MTYPE,
   2,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
        | ELEMENT_PROPAGATE,
    TPMI_ALG_SIG_SCHEME_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_SIGNATURE, sigAlg)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
    TPMU_SIGNATURE_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_SIGNATURE, signature))}},
  // TPMU_ENCRYPTED_SECRET_DATA
  {4,
   IS_ARRAY_UNION,
   (UINT16)(offsetof(TPMU_ENCRYPTED_SECRET_mst, marshalingTypes)),
   {(UINT32)TPM_ALG_ECC,
    (UINT32)TPM_ALG_RSA,
    (UINT32)TPM_ALG_SYMCIPHER,
    (UINT32)TPM_ALG_KEYEDHASH},
   {(UINT16)(sizeof(TPMS_ECC_POINT)),
    (UINT16)(MAX_RSA_KEY_BYTES),
    (UINT16)(sizeof(TPM2B_DIGEST)),
    (UINT16)(sizeof(TPM2B_DIGEST))}},
  // TPM2B_ENCRYPTED_SECRET_DATA
  {TPM2B_MTYPE, Type36_MARSHAL_REF},
  // TPMI_ALG_PUBLIC_DATA
  {MIN_MAX_MTYPE,
   TWO_BYTES | HAS_BITS,
   (UINT8)TPM_RC_TYPE,
   {RANGE(1, 37, UINT16),
    (UINT32)((ALG_RSA << 0) | (ALG_KEYEDHASH << 7)),
    (UINT32)((ALG_ECC << 2) | (ALG_SYMCIPHER << 4))}},
  // TPMU_PUBLIC_ID_DATA
  {4,
   0,
   (UINT16)(offsetof(TPMU_PUBLIC_ID_mst, marshalingTypes)),
   {(UINT32)TPM_ALG_KEYEDHASH,
    (UINT32)TPM_ALG_SYMCIPHER,
    (UINT32)TPM_ALG_RSA,
    (UINT32)TPM_ALG_ECC},
   {(UINT16)(TPM2B_DIGEST_MARSHAL_REF),
    (UINT16)(TPM2B_DIGEST_MARSHAL_REF),
    (UINT16)(TPM2B_PUBLIC_KEY_RSA_MARSHAL_REF),
    (UINT16)(TPMS_ECC_POINT_MARSHAL_REF)}},
  // TPMS_KEYEDHASH_PARMS_DATA
  {STRUCTURE_MTYPE,
   1,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_KEYEDHASH_SCHEME_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_KEYEDHASH_PARMS, scheme))}},
  // TPMS_RSA_PARMS_DATA
  {STRUCTURE_MTYPE,
   4,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_SYM_DEF_OBJECT_MARSHAL_REF | NULL_FLAG,
```

```
    (UINT16)(offsetof(TPMS_RSA_PARMS, symmetric)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_RSA_SCHEME_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_RSA_PARMS, scheme)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_RSA_KEY_BITS_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_RSA_PARMS, keyBits)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
    UINT32_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_RSA_PARMS, exponent))}},
  // TPMS_ECC_PARMS_DATA
  {STRUCTURE_MTYPE,
   4,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_SYM_DEF_OBJECT_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_ECC_PARMS, symmetric)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_ECC_SCHEME_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_ECC_PARMS, scheme)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ECC_CURVE_MARSHAL_REF,
    (UINT16)(offsetof(TPMS_ECC_PARMS, curveID)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPMT_KDF_SCHEME_MARSHAL_REF | NULL_FLAG,
    (UINT16)(offsetof(TPMS_ECC_PARMS, kdf))}},
  // TPMU_PUBLIC_PARMS_DATA
  {4,
   0,
   (UINT16)(offsetof(TPMU_PUBLIC_PARMS_mst, marshalingTypes)),
   {(UINT32)TPM_ALG_KEYEDHASH,
    (UINT32)TPM_ALG_SYMCIPHER,
    (UINT32)TPM_ALG_RSA,
    (UINT32)TPM_ALG_ECC},
   {(UINT16)(TPMS_KEYEDHASH_PARMS_MARSHAL_REF),
    (UINT16)(TPMS_SYMCIPHER_PARMS_MARSHAL_REF),
    (UINT16)(TPMS_RSA_PARMS_MARSHAL_REF),
    (UINT16)(TPMS_ECC_PARMS_MARSHAL_REF)}},
  // TPMT_PUBLIC_PARMS_DATA
  {STRUCTURE_MTYPE,
   2,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ALG_PUBLIC_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_PUBLIC_PARMS, type)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
    TPMU_PUBLIC_PARMS_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_PUBLIC_PARMS, parameters))}},
  // TPMT_PUBLIC_DATA
  {STRUCTURE_MTYPE,
   6,
   {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
    TPMI_ALG_PUBLIC_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_PUBLIC, type)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES)
        | ELEMENT_PROPAGATE,
    TPMI_ALG_HASH_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_PUBLIC, nameAlg)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
    TPMA_OBJECT_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_PUBLIC, objectAttributes)),
    SET_ELEMENT_TYPE(SIMPLE_STYPE),
    TPM2B_DIGEST_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_PUBLIC, authPolicy)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
    TPMU_PUBLIC_PARMS_MARSHAL_REF,
    (UINT16)(offsetof(TPMT_PUBLIC, parameters)),
    SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
```

```
  TPMU_PUBLIC_ID_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_PUBLIC, unique))},
// TPM2B_PUBLIC_DATA
{TPM2BS_MTYPE,
 (UINT8)(offsetof(TPM2B_PUBLIC, publicArea)) | SIZE_EQUAL | ELEMENT_PROPAGATE,
 UINT16_MARSHAL_REF,
 TPMT_PUBLIC_MARSHAL_REF},
// TPM2B_TEMPLATE_DATA
{TPM2B_MTYPE, Type37_MARSHAL_REF},
// TPM2B_PRIVATE_VENDOR_SPECIFIC_DATA
{TPM2B_MTYPE, Type38_MARSHAL_REF},
// TPMU_SENSITIVE_COMPOSITE_DATA
{4,
 0,
 (UINT16)(offsetof(TPMU_SENSITIVE_COMPOSITE_mst, marshalingTypes)),
 {(UINT32)TPM_ALG_RSA,
  (UINT32)TPM_ALG_ECC,
  (UINT32)TPM_ALG_KEYEDHASH,
  (UINT32)TPM_ALG_SYMCIPHER},
 {(UINT16)(TPM2B_PRIVATE_KEY_RSA_MARSHAL_REF),
  (UINT16)(TPM2B_ECC_PARAMETER_MARSHAL_REF),
  (UINT16)(TPM2B_SENSITIVE_DATA_MARSHAL_REF),
  (UINT16)(TPM2B_SYM_KEY_MARSHAL_REF)}},
// TPMT_SENSITIVE_DATA
{STRUCTURE_MTYPE,
 4,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPMI_ALG_PUBLIC_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SENSITIVE, sensitiveType)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_AUTH_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SENSITIVE, authValue)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SENSITIVE, seedValue)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_SENSITIVE_COMPOSITE_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_SENSITIVE, sensitive))}},
// TPM2B_SENSITIVE_DATA
{TPM2BS_MTYPE,
 (UINT8)(offsetof(TPM2B_SENSITIVE, sensitiveArea)),
 UINT16_MARSHAL_REF,
 TPMT_SENSITIVE_MARSHAL_REF},
// TPM2B_PRIVATE_DATA
{TPM2B_MTYPE, Type39_MARSHAL_REF},
// TPM2B_ID_OBJECT_DATA
{TPM2B_MTYPE, Type40_MARSHAL_REF},
// TPMS_NV_PIN_COUNTER_PARAMETERS_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  UINT32_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PIN_COUNTER_PARAMETERS, pinCount)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  UINT32_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PIN_COUNTER_PARAMETERS, pinLimit))}},
// TPMA_NV_DATA
{ATTRIBUTES_MTYPE, FOUR_BYTES, 0x01F00300},
// TPMA_NV_EXP_DATA
{ATTRIBUTES_MTYPE, EIGHT_BYTES, 0xffffff801F00300},
// TPMS_NV_PUBLIC_DATA
{STRUCTURE_MTYPE,
 5,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMI_RH_NV_INDEX_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC, nvIndex)),
```

```
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPMI_ALG_HASH_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC, nameAlg)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMA_NV_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC, attributes)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC, authPolicy)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  Type41_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC, dataSize))}},
// TPM2B_NV_PUBLIC_DATA
{TPM2BS_MTYPE,
 (UINT8)(offsetof(TPM2B_NV_PUBLIC, nvPublic)) | SIZE_EQUAL,
 UINT16_MARSHAL_REF,
 TPMS_NV_PUBLIC_MARSHAL_REF},
// TPMS_NV_PUBLIC_EXP_ATTR_DATA
{STRUCTURE_MTYPE,
 5,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPMI_RH_NV_EXP_INDEX_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC_EXP_ATTR, nvIndex)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  TPMI_ALG_HASH_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC_EXP_ATTR, nameAlg)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(EIGHT_BYTES),
  TPMA_NV_EXP_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC_EXP_ATTR, attributes)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC_EXP_ATTR, authPolicy)),
  SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
  Type41_MARSHAL_REF,
  (UINT16)(offsetof(TPMS_NV_PUBLIC_EXP_ATTR, dataSize))}},
// TPMU_NV_PUBLIC_2_DATA
{3,
 0,
 (UINT16)(offsetof(TPMU_NV_PUBLIC_2_mst, marshalingTypes)),
 {(UINT32)TPM_HT_NV_INDEX,
  (UINT32)TPM_HT_EXTERNAL_NV,
  (UINT32)TPM_HT_PERMANENT_NV},
 {(UINT16)(TPMS_NV_PUBLIC_MARSHAL_REF),
  (UINT16)(TPMS_NV_PUBLIC_EXP_ATTR_MARSHAL_REF),
  (UINT16)(TPMS_NV_PUBLIC_MARSHAL_REF)}},
// TPMT_NV_PUBLIC_2_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
  TPM_HT_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_NV_PUBLIC_2, handleType)),
  SET_ELEMENT_TYPE(UNION_STYPE) | SET_ELEMENT_NUMBER(0),
  TPMU_NV_PUBLIC_2_MARSHAL_REF,
  (UINT16)(offsetof(TPMT_NV_PUBLIC_2, nvPublic2))}},
// TPM2B_NV_PUBLIC_2_DATA
{TPM2BS_MTYPE,
 (UINT8)(offsetof(TPM2B_NV_PUBLIC_2, nvPublic2)) | SIZE_EQUAL,
 UINT16_MARSHAL_REF,
 TPMT_NV_PUBLIC_2_MARSHAL_REF},
// TPM2B_CONTEXT_SENSITIVE_DATA
{TPM2B_MTYPE, Type42_MARSHAL_REF},
// TPMS_CONTEXT_DATA_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPM2B_DIGEST_MARSHAL_REF,
```

```
   (UINT16)(offsetof(TPMS_CONTEXT_DATA, integrity)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE),
   TPM2B_CONTEXT_SENSITIVE_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CONTEXT_DATA, encrypted))}},
// TPM2B_CONTEXT_DATA_DATA
{TPM2B_MTYPE, Type43_MARSHAL_REF},
// TPMS_CONTEXT_DATA
{STRUCTURE_MTYPE,
 4,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(EIGHT_BYTES),
  UINT64_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CONTEXT, sequence)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
   TPMI_DH_SAVED_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CONTEXT, savedHandle)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
   TPMI_RH_HIERARCHY_MARSHAL_REF | NULL_FLAG,
   (UINT16)(offsetof(TPMS_CONTEXT, hierarchy)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE),
   TPM2B_CONTEXT_DATA_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CONTEXT, contextBlob))}},
// TPMS_CREATION_DATA_DATA
{STRUCTURE_MTYPE,
 7,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE),
  TPML_PCR_SELECTION_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CREATION_DATA, pcrSelect)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE),
   TPM2B_DIGEST_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CREATION_DATA, pcrDigest)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(ONE_BYTES),
   TPMA_LOCALITY_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CREATION_DATA, locality)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(TWO_BYTES),
   TPM_ALG_ID_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CREATION_DATA, parentNameAlg)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE),
   TPM2B_NAME_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CREATION_DATA, parentName)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE),
   TPM2B_NAME_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CREATION_DATA, parentQualifiedName)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE),
   TPM2B_DATA_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_CREATION_DATA, outsideInfo))}},
// TPM2B_CREATION_DATA_DATA
{TPM2BS_MTYPE,
 (UINT8)(offsetof(TPM2B_CREATION_DATA, creationData)) | SIZE_EQUAL,
 UINT16_MARSHAL_REF,
 TPMS_CREATION_DATA_MARSHAL_REF},
// TPM_AT_DATA
{TABLE_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_VALUE,
 4,
 {TPM_AT_ANY, TPM_AT_ERROR, TPM_AT_PV1, TPM_AT_VEND}},
// TPMS_AC_OUTPUT_DATA
{STRUCTURE_MTYPE,
 2,
 {SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
  TPM_AT_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_AC_OUTPUT, tag)),
   SET_ELEMENT_TYPE(SIMPLE_STYPE) | SET_ELEMENT_SIZE(FOUR_BYTES),
   UINT32_MARSHAL_REF,
   (UINT16)(offsetof(TPMS_AC_OUTPUT, data))}},
// TPML_AC_CAPABILITIES_DATA
```

```
{LIST_MTYPE,
 (UINT8)(offsetof(TPML_AC_CAPABILITIES, acCapabilities)),
 Type44_MARSHAL_REF,
 TPMS_AC_OUTPUT_ARRAY_MARSHAL_INDEX},
// Type00_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, sizeof(TPMU_HA), UINT16)}},
// Type01_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, sizeof(TPMT_HA), UINT16)}},
// Type02_DATA
{MIN_MAX_MTYPE, TWO_BYTES, (UINT8)TPM_RC_SIZE, {RANGE(0, 1024, UINT16)}},
// Type03_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, MAX_DIGEST_BUFFER, UINT16)}},
// Type04_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, MAX_NV_BUFFER_SIZE, UINT16)}},
// Type05_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, sizeof(UINT64), UINT16)}},
// Type06_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, MAX_SYM_BLOCK_SIZE, UINT16)}},
// Type07_DATA
{MIN_MAX_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, sizeof(TPMU_NAME), UINT16)}},
// Type08_DATA
{MIN_MAX_MTYPE,
 ONE_BYTES,
 (UINT8)TPM_RC_VALUE,
 {RANGE(PCR_SELECT_MIN, PCR_SELECT_MAX, UINT8)}},
// Type10_DATA
{TABLE_MTYPE, TWO_BYTES, (UINT8)TPM_RC_TAG, 1, {TPM_ST_CREATION}},
// Type11_DATA
{TABLE_MTYPE, TWO_BYTES, (UINT8)TPM_RC_TAG, 1, {TPM_ST_VERIFIED}},
// Type12_DATA
{TABLE_MTYPE,
 TWO_BYTES,
 (UINT8)TPM_RC_TAG,
 2,
 {TPM_ST_AUTH_SECRET, TPM_ST_AUTH_SIGNED}},
// Type13_DATA
{TABLE_MTYPE, TWO_BYTES, (UINT8)TPM_RC_TAG, 1, {TPM_ST_HASHCHECK}},
// Type15_DATA
{MIN_MAX_MTYPE, FOUR_BYTES, (UINT8)TPM_RC_SIZE, {RANGE(0, MAX_CAP_CC, UINT32)}},
// Type17_DATA
{MIN_MAX_MTYPE,
 FOUR_BYTES,
 (UINT8)TPM_RC_SIZE,
 {RANGE(0, MAX_ALG_LIST_SIZE, UINT32)}},
// Type18_DATA
```

```
    {MIN_MAX_MTYPE,
     FOUR_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_CAP_HANDLES, UINT32)}},
    // Type19_DATA
    {MIN_MAX_MTYPE, FOUR_BYTES, (UINT8)TPM_RC_SIZE, {RANGE(2, 8, UINT32)}},
    // Type20_DATA
    {MIN_MAX_MTYPE, FOUR_BYTES, (UINT8)TPM_RC_SIZE, {RANGE(0, HASH_COUNT, UINT32)}},
    // Type22_DATA
    {MIN_MAX_MTYPE, FOUR_BYTES, (UINT8)TPM_RC_SIZE, {RANGE(0, MAX_CAP_ALGS, UINT32)}},
    // Type23_DATA
    {MIN_MAX_MTYPE,
     FOUR_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_TPM_PROPERTIES, UINT32)}},
    // Type24_DATA
    {MIN_MAX_MTYPE,
     FOUR_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_PCR_PROPERTIES, UINT32)}},
    // Type25_DATA
    {MIN_MAX_MTYPE,
     FOUR_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_ECC_CURVES, UINT32)}},
    // Type26_DATA
    {MIN_MAX_MTYPE,
     FOUR_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_TAGGED_POLICIES, UINT32)}},
    // Type27_DATA
    {MIN_MAX_MTYPE, FOUR_BYTES, (UINT8)TPM_RC_SIZE, {RANGE(0, MAX_ACT_DATA, UINT32)}},
    // Type28_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(TPMS_ATTEST), UINT16)}},
    // Type29_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_SYM_KEY_BYTES, UINT16)}},
    // Type30_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, LABEL_MAX_BUFFER, UINT16)}},
    // Type31_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(TPMS_DERIVE), UINT16)}},
    // Type32_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(TPMU_SENSITIVE_CREATE), UINT16)}},
    // Type33_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_RSA_KEY_BYTES, UINT16)}},
    // Type34_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
```

```
     {RANGE(0, RSA_PRIVATE_SIZE, UINT16)}},
    // Type35_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_ECC_KEY_BYTES, UINT16)}},
    // Type36_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(TPMU_ENCRYPTED_SECRET), UINT16)}},
    // Type37_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(TPMT_PUBLIC), UINT16)}},
    // Type38_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, PRIVATE_VENDOR_SPECIFIC_BYTES, UINT16)}},
    // Type39_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(_PRIVATE), UINT16)}},
    // Type40_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(TPMS_ID_OBJECT), UINT16)}},
    // Type41_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_NV_INDEX_SIZE, UINT16)}},
    // Type42_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_CONTEXT_SIZE, UINT16)}},
    // Type43_DATA
    {MIN_MAX_MTYPE,
     TWO_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, sizeof(TPMS_CONTEXT_DATA), UINT16)}},
    // Type44_DATA
    {MIN_MAX_MTYPE,
     FOUR_BYTES,
     (UINT8)TPM_RC_SIZE,
     {RANGE(0, MAX_AC_CAPABILITIES, UINT32)}}};
#endif  // TABLE_DRIVEN_MARSHAL
```

7.198 /tpm/src/support/TpmFail.c

```
//** Includes, Defines, and Types
#define TPM_FAIL_C
#include "Tpm.h"

// On MS C compiler, can save the alignment state and set the alignment to 1 for
// the duration of the TpmTypes.h include.  This will avoid a lot of alignment
// warnings from the compiler for the unaligned structures. The alignment of the
// structures is not important as this function does not use any of the structures
// in TpmTypes.h and only include it for the #defines of the capabilities,
// properties, and command code values.
```

```
#include "TpmTypes.h"

//** Typedefs
// These defines are used primarily for sizing of the local response buffer.
typedef struct
{
    TPM_ST tag;
    UINT32 size;
    TPM_RC code;
} HEADER;

typedef struct
{
    BYTE tag[sizeof(TPM_ST)];
    BYTE size[sizeof(UINT32)];
    BYTE code[sizeof(TPM_RC)];
} PACKED_HEADER;

typedef struct
{
    BYTE size[sizeof(UINT16)];
    struct
    {
        BYTE function[sizeof(UINT32)];
        BYTE line[sizeof(UINT32)];
        BYTE code[sizeof(UINT32)];
    } values;
    BYTE returnCode[sizeof(TPM_RC)];
} GET_TEST_RESULT_PARAMETERS;

typedef struct
{
    BYTE moreData[sizeof(TPMI_YES_NO)];
    BYTE capability[sizeof(TPM_CAP)];  // Always TPM_CAP_TPM_PROPERTIES
    BYTE tpmProperty[sizeof(TPML_TAGGED_TPM_PROPERTY)];
} GET_CAPABILITY_PARAMETERS;

typedef struct
{
    BYTE header[sizeof(PACKED_HEADER)];
    BYTE getTestResult[sizeof(GET_TEST_RESULT_PARAMETERS)];
} TEST_RESPONSE;

typedef struct
{
    BYTE header[sizeof(PACKED_HEADER)];
    BYTE getCap[sizeof(GET_CAPABILITY_PARAMETERS)];
} CAPABILITY_RESPONSE;

typedef union
{
    BYTE test[sizeof(TEST_RESPONSE)];
    BYTE cap[sizeof(CAPABILITY_RESPONSE)];
} RESPONSES;

// Buffer to hold the responses. This may be a little larger than
// required due to padding that a compiler might add.
// Note: This is not in Global.c because of the specialized data definitions above.
// Since the data contained in this structure is not relevant outside of the
// execution of a single command (when the TPM is in failure mode. There is no
// compelling reason to move all the typedefs to Global.h and this structure
// to Global.c.
#ifndef __IGNORE_STATE__  // Don't define this value
static BYTE response[sizeof(RESPONSES)];
#endif
```

```
//** Local Functions

//*** MarshalUint16()
// Function to marshal a 16 bit value to the output buffer.
static INT32 MarshalUint16(UINT16 integer, BYTE** buffer)
{
    UINT16_TO_BYTE_ARRAY(integer, *buffer);
    *buffer += 2;
    return 2;
}

//*** MarshalUint32()
// Function to marshal a 32 bit value to the output buffer.
static INT32 MarshalUint32(UINT32 integer, BYTE** buffer)
{
    UINT32_TO_BYTE_ARRAY(integer, *buffer);
    *buffer += 4;
    return 4;
}

//***Unmarshal32()
static BOOL Unmarshal32(UINT32* target, BYTE** buffer, INT32* size)
{
    if((*size -= 4) < 0)
        return FALSE;
    *target = BYTE_ARRAY_TO_UINT32(*buffer);
    *buffer += 4;
    return TRUE;
}

//***Unmarshal16()
static BOOL Unmarshal16(UINT16* target, BYTE** buffer, INT32* size)
{
    if((*size -= 2) < 0)
        return FALSE;
    *target = BYTE_ARRAY_TO_UINT16(*buffer);
    *buffer += 2;
    return TRUE;
}

//** Public Functions

//*** SetForceFailureMode()
// This function is called by the simulator to enable failure mode testing.
#if ALLOW_FORCE_FAILURE_MODE
LIB_EXPORT void SetForceFailureMode(void)
{
    g_forceFailureMode = TRUE;
    return;
}
#endif  // ALLOW_FORCE_FAILURE_MODE

//*** TpmFail()
// This function is called by TPM.lib when a failure occurs. It will set up the
// failure values to be returned on TPM2_GetTestResult().
NORETURN void TpmFail(
#if FAIL_TRACE
    const char* function,
    int         line,
#else
    uint64_t locationCode,
#endif
    int failureCode)
{
    // Save the values that indicate where the error occurred.
    // On a 64-bit machine, this may truncate the address of the string
```

```
    // of the function name where the error occurred.
#if FAIL_TRACE
    s_failFunctionName = function;
    s_failFunction     = (UINT32)(ptrdiff_t)function;
    s_failLine         = line;
#else
    s_failFunction = (UINT32)(locationCode >> 32);
    s_failLine     = (UINT32)(locationCode);
#endif
    s_failCode = failureCode;

    // We are in failure mode
    g_inFailureMode = TRUE;

    // Notify the platform that we hit a failure.
    //
    // In the LONGJMP case, the reference platform code is expected to long-jmp
    // back to the ExecuteCommand call and output a failure response.
    //
    // In the NO_LONGJMP case, this is a notification to the platform, and the
    // platform may take any (implementation-defined) behavior, including no-op,
    // debugging, or whatever. The core library is expected to surface the failure
    // back to ExecuteCommand through error propagation and return an appropriate
    // failure reply.
    _plat__Fail();
}

//*** TpmFailureMode(
// This function is called by the interface code when the platform is in failure
// mode.
void TpmFailureMode(uint32_t        inRequestSize,    // IN: command buffer size
                    unsigned char*  inRequest,        // IN: command buffer
                    uint32_t*       outResponseSize,  // OUT: response buffer size
                    unsigned char** outResponse       // OUT: response buffer
)
{
    UINT32 marshalSize;
    UINT32 capability;
    HEADER header;  // unmarshaled command header
    UINT32 pt;      // unmarshaled property type
    UINT32 count;   // unmarshaled property count
    UINT8* buffer = inRequest;
    INT32  size   = inRequestSize;

    // If there is no command buffer, then just return TPM_RC_FAILURE
    if(inRequestSize == 0 || inRequest == NULL)
        goto FailureModeReturn;
    // If the header is not correct for TPM2_GetCapability() or
    // TPM2_GetTestResult() then just return the in failure mode response;
    if(!(Unmarshal16(&header.tag, &buffer, &size)
         && Unmarshal32(&header.size, &buffer, &size)
         && Unmarshal32(&header.code, &buffer, &size)))
        goto FailureModeReturn;
    if(header.tag != TPM_ST_NO_SESSIONS || header.size < 10)
        goto FailureModeReturn;
    switch(header.code)
    {
        case TPM_CC_GetTestResult:
            // make sure that the command size is correct
            if(header.size != 10)
                goto FailureModeReturn;
            buffer      = &response[10];
            marshalSize = MarshalUint16(3 * sizeof(UINT32), &buffer);
            marshalSize += MarshalUint32(s_failFunction, &buffer);
            marshalSize += MarshalUint32(s_failLine, &buffer);
            marshalSize += MarshalUint32(s_failCode, &buffer);
```

```
        if(s_failCode == FATAL_ERROR_NV_UNRECOVERABLE)
            marshalSize += MarshalUint32(TPM_RC_NV_UNINITIALIZED, &buffer);
        else
            marshalSize += MarshalUint32(TPM_RC_FAILURE, &buffer);
        break;
    case TPM_CC_GetCapability:
        // make sure that the size of the command is exactly the size
        // returned for the capability, property, and count
        if(header.size != (10 + (3 * sizeof(UINT32)))
           // also verify that this is requesting TPM properties
           || !Unmarshal32(&capability, &buffer, &size)
           || capability != TPM_CAP_TPM_PROPERTIES
           || !Unmarshal32(&pt, &buffer, &size)
           || !Unmarshal32(&count, &buffer, &size))
            goto FailureModeReturn;

        if(count > 0)
            count = 1;
        else if(pt > TPM_PT_FIRMWARE_VERSION_2)
            count = 0;
        if(pt < TPM_PT_MANUFACTURER)
            pt = TPM_PT_MANUFACTURER;
        // set up for return
        buffer = &response[10];
        // if the request was for a PT less than the last one
        // then we indicate more, otherwise, not.
        if(pt < TPM_PT_FIRMWARE_VERSION_2)
            *buffer++ = YES;
        else
            *buffer++ = NO;
        marshalSize = 1;

        // indicate the capability type
        marshalSize += MarshalUint32(capability, &buffer);
        // indicate the number of values that are being returned (0 or 1)
        marshalSize += MarshalUint32(count, &buffer);
        // indicate the property
        marshalSize += MarshalUint32(pt, &buffer);

        if(count > 0)
            switch(pt)
            {
                case TPM_PT_MANUFACTURER:
                    // the vendor ID unique to each TPM manufacturer
                    pt = _plat__GetManufacturerCapabilityCode();
                    break;

                case TPM_PT_VENDOR_STRING_1:
                    // the first four characters of the vendor ID string
                    pt = _plat__GetVendorCapabilityCode(1);
                    break;

                case TPM_PT_VENDOR_STRING_2:
                    // the second four characters of the vendor ID string
                    pt = _plat__GetVendorCapabilityCode(2);
                    break;

                case TPM_PT_VENDOR_STRING_3:
                    // the third four characters of the vendor ID string
                    pt = _plat__GetVendorCapabilityCode(3);
                    break;

                case TPM_PT_VENDOR_STRING_4:
                    // the fourth four characters of the vendor ID string
                    pt = _plat__GetVendorCapabilityCode(4);
                    break;
```

```
                    case TPM_PT_VENDOR_TPM_TYPE:
                        // vendor-defined value indicating the TPM model
                        // We just make up a number here
                        pt = _plat__GetTpmType();
                        break;

                    case TPM_PT_FIRMWARE_VERSION_1:
                        // the more significant 32-bits of a vendor-specific value
                        // indicating the version of the firmware
                        pt = _plat__GetTpmFirmwareVersionHigh();
                        break;

                    default:  // TPM_PT_FIRMWARE_VERSION_2:
                        // the less significant 32-bits of a vendor-specific value
                        // indicating the version of the firmware
                        pt = _plat__GetTpmFirmwareVersionLow();
                        break;
                }
            marshalSize += MarshalUint32(pt, &buffer);
            break;
        default:  // default for switch (cc)
            goto FailureModeReturn;
    }
    // Now do the header
    buffer      = response;
    marshalSize = marshalSize + 10;             // Add the header size to the
                                                // stuff already marshaled
    MarshalUint16(TPM_ST_NO_SESSIONS, &buffer); // structure tag
    MarshalUint32(marshalSize, &buffer);        // responseSize
    MarshalUint32(TPM_RC_SUCCESS, &buffer);     // response code

    *outResponseSize = marshalSize;
    *outResponse     = (unsigned char*)&response;
    return;
FailureModeReturn:
    buffer      = response;
    marshalSize = MarshalUint16(TPM_ST_NO_SESSIONS, &buffer);
    marshalSize += MarshalUint32(10, &buffer);
    marshalSize += MarshalUint32(TPM_RC_FAILURE, &buffer);
    *outResponseSize = marshalSize;
    *outResponse     = (unsigned char*)response;
    return;
}

//*** UnmarshalFail()
// This is a stub that is used to catch an attempt to unmarshal an entry
// that is not defined. Don't ever expect this to be called but...
void UnmarshalFail(void* type, BYTE** buffer, INT32* size)
{
    NOT_REFERENCED(type);
    NOT_REFERENCED(buffer);
    NOT_REFERENCED(size);
    FAIL(FATAL_ERROR_INTERNAL);
}
```

7.199 /tpm/src/support/TpmSizeChecks.c

```
//** Includes, Defines, and Types
#include "Tpm.h"
#include <stdio.h>
#include <assert.h>
#include "Marshal.h"

#if RUNTIME_SIZE_CHECKS
```

```
#  if DEBUG
static int once = 0;
#  endif

//** TpmSizeChecks()
// This function is used during the development process to make sure that the
// vendor-specific values result in a consistent implementation. When possible,
// the code contains #if to do compile-time checks. However, in some cases, the
// values require the use of "sizeof()" and that can't be used in an #if.
BOOL TpmSizeChecks(void)
{
    BOOL PASS = TRUE;

#  if DEBUG
    //
    if(once++ != 0)
        return 1;

#    if ALG_ECC
    {
        // This is just to allow simple access to the ecc curve data during debug
        const TPM_ECC_CURVE_METADATA* ecc = CryptEccGetParametersByCurveId(3);
        if(ecc == NULL)
            ecc = NULL;
    }
#    endif  // ALG_ECC
    {
        UINT32 maxAsymSecurityStrength  = MAX_ASYM_SECURITY_STRENGTH;
        UINT32 maxHashSecurityStrength  = MAX_HASH_SECURITY_STRENGTH;
        UINT32 maxSymSecurityStrength   = MAX_SYM_SECURITY_STRENGTH;
        UINT32 maxSecurityStrengthBits  = MAX_SECURITY_STRENGTH_BITS;
        UINT32 proofSize                = PROOF_SIZE;
        UINT32 compliantProofSize       = COMPLIANT_PROOF_SIZE;
        UINT32 compliantPrimarySeedSize = COMPLIANT_PRIMARY_SEED_SIZE;
        UINT32 primarySeedSize          = PRIMARY_SEED_SIZE;

        UINT32 cmacState                = sizeof(tpmCmacState_t);
        UINT32 hashState                = sizeof(HASH_STATE);
        UINT32 keyScheduleSize          = sizeof(tpmCryptKeySchedule_t);
        //
        NOT_REFERENCED(cmacState);
        NOT_REFERENCED(hashState);
        NOT_REFERENCED(keyScheduleSize);
        NOT_REFERENCED(maxAsymSecurityStrength);
        NOT_REFERENCED(maxHashSecurityStrength);
        NOT_REFERENCED(maxSymSecurityStrength);
        NOT_REFERENCED(maxSecurityStrengthBits);
        NOT_REFERENCED(proofSize);
        NOT_REFERENCED(compliantProofSize);
        NOT_REFERENCED(compliantPrimarySeedSize);
        NOT_REFERENCED(primarySeedSize);

#    if ALG_RSA
        {
            TPMT_SENSITIVE* p;
            // This assignment keeps compiler from complaining about a conditional
            // comparison being between two constants
            UINT16 max_rsa_key_bytes = MAX_RSA_KEY_BYTES;
            if((max_rsa_key_bytes / 2) != (sizeof(p->sensitive.rsa.t.buffer) / 5))
            {
                printf("Sensitive part of TPMT_SENSITIVE is undersized. May be "
                       "caused"
                       " by use of wrong version of Part 2.\n");
                PASS = FALSE;
            }
```

```
        }
#    endif  // ALG_RSA
#    if TABLE_DRIVEN_MARSHAL
        printf("sizeof(MarshalData) = %zu\n", sizeof(MarshalData_st));
#    endif

        printf("Size of OBJECT = %zu\n", sizeof(OBJECT));
        printf("Size of components in TPMT_SENSITIVE = %zu\n",
               sizeof(TPMT_SENSITIVE));
        printf("    TPMI_ALG_PUBLIC                 %zu\n", sizeof(TPMI_ALG_PUBLIC));
        printf("    TPM2B_AUTH                      %zu\n", sizeof(TPM2B_AUTH));
        printf("    TPM2B_DIGEST                    %zu\n", sizeof(TPM2B_DIGEST));
        printf("    TPMU_SENSITIVE_COMPOSITE        %zu\n",
               sizeof(TPMU_SENSITIVE_COMPOSITE));
    }
    // Make sure that the size of the context blob is large enough for the largest
    // context
    // TPMS_CONTEXT_DATA contains two TPM2B values. That is not how this is
    // implemented. Rather, the size field of the TPM2B_CONTEXT_DATA is used to
    // determine the amount of data in the encrypted data. That part is not
    // independently sized. This makes the actual size 2 bytes smaller than
    // calculated using Part 2. Since this is opaque to the caller, it is not
    // necessary to fix. The actual size is returned by TPM2_GetCapabilties().

    // Initialize output handle.  At the end of command action, the output
    // handle of an object will be replaced, while the output handle
    // for a session will be the same as input

    // Get the size of fingerprint in context blob.  The sequence value in
    // TPMS_CONTEXT structure is used as the fingerprint
    {
        UINT32 fingerprintSize = sizeof(UINT64);
        UINT32 integritySize =
            sizeof(UINT16) + CryptHashGetDigestSize(CONTEXT_INTEGRITY_HASH_ALG);
        UINT32 biggestObject =
            MAX(MAX(sizeof(HASH_OBJECT), sizeof(OBJECT)), sizeof(SESSION));
        UINT32 biggestContext = fingerprintSize + integritySize + biggestObject;

        // round required size up to nearest 8 byte boundary.
        biggestContext = 8 * ((biggestContext + 7) / 8);

        if(MAX_CONTEXT_SIZE < biggestContext)
        {
            printf("MAX_CONTEXT_SIZE needs to be increased to at least %d (%d)\n",
                   biggestContext,
                   MAX_CONTEXT_SIZE);
            PASS = FALSE;
        }
        else if(MAX_CONTEXT_SIZE > biggestContext)
        {
            printf("MAX_CONTEXT_SIZE can be reduced to %d (%d)\n",
                   biggestContext,
                   MAX_CONTEXT_SIZE);
        }
    }
    {
        union u
        {
            TPMA_OBJECT attributes;
            UINT32      uint32Value;
        } u;
        // these are defined so that compiler doesn't complain about conditional
        // expressions comparing two constants.
        int aSize     = sizeof(u.attributes);
        int uSize     = sizeof(u.uint32Value);
        u.uint32Value = 0;
```

```
        SET_ATTRIBUTE(u.attributes, TPMA_OBJECT, fixedTPM);
        if(u.uint32Value != 2)
        {
            printf("The bit allocation in a TPMA_OBJECT is not as expected");
            PASS = FALSE;
        }
        if(aSize != uSize)  // comparison of two sizeof() values annoys compiler
        {
            printf("A TPMA_OBJECT is not the expected size.");
            PASS = FALSE;
        }
    }
#    if ACT_SUPPORT
    // Check that the platorm implementes each of the ACT that the TPM thinks are
    // present
    {
        uint32_t act;
        for(act = 0; act < 16; act++)
        {
            switch(act)
            {
                FOR_EACH_ACT(CASE_ACT_NUMBER)
                if(!_plat__ACT_GetImplemented(act))
                {
                    printf("TPM_RH_ACT_%1X is not implemented by platform\n", act);
                    PASS = FALSE;
                }
                default:
                    break;
            }
        }
    }
#    endif  // ACT_SUPPORT
    {
        // Had a problem with the macros coming up with some bad values. Make sure
        // the size is rational
        int t = MAX_DIGEST_SIZE;
        if(t < 20)
        {
            printf("Check the MAX_DIGEST_SIZE computation (%d)", MAX_DIGEST_SIZE);
            PASS = FALSE;
        }
    }
#  endif  // DEBUG
    return (PASS);
}

#endif  // RUNTIME_SIZE_CHECKS
```

7.200 /tpm/src/X509/TpmASN1.c

```
//** Includes
#include "Tpm.h"
#define _OIDS_
#include "OIDs.h"
#include "TpmASN1.h"
#include "TpmASN1_fp.h"

#if CC_CertifyX509

//** Unmarshaling Functions

//*** ASN1UnmarshalContextInitialize()
// Function does standard initialization of a context.
//  Return Type: BOOL
```

```
//      TRUE(1)     success
//      FALSE(0)    failure
BOOL ASN1UnmarshalContextInitialize(
    ASN1UnmarshalContext* ctx, INT16 size, BYTE* buffer)
{
    GOTO_ERROR_UNLESS(buffer != NULL);
    GOTO_ERROR_UNLESS(size > 0);
    ctx->buffer = buffer;
    ctx->size   = size;
    ctx->offset = 0;
    ctx->tag    = 0xFF;
    return TRUE;
Error:
    return FALSE;
}

//***ASN1DecodeLength()
// This function extracts the length of an element from 'buffer' starting at 'offset'.
// Return Type: UINT16
//      >=0         the extracted length
//      <0          an error
INT16
ASN1DecodeLength(ASN1UnmarshalContext* ctx)
{
    BYTE  first;  // Next octet in buffer
    INT16 value;
    //
    GOTO_ERROR_UNLESS(ctx->offset < ctx->size);
    first = NEXT_OCTET(ctx);
    // If the number of octets of the entity is larger than 127, then the first octet
    // is the number of octets in the length specifier.
    if(first >= 0x80)
    {
        // Make sure that this length field is contained with the structure being
        // parsed
        CHECK_SIZE(ctx, (first & 0x7F));
        if(first == 0x82)
        {
            // Two octets of size
            // get the next value
            value = (INT16)NEXT_OCTET(ctx);
            // Make sure that the result will fit in an INT16
            GOTO_ERROR_UNLESS(value < 0x0080);
            // Shift up and add next octet
            value = (value << 8) + NEXT_OCTET(ctx);
        }
        else if(first == 0x81)
            value = NEXT_OCTET(ctx);
        // Sizes larger than will fit in a INT16 are an error
        else
            goto Error;
    }
    else
        value = first;
    // Make sure that the size defined something within the current context
    CHECK_SIZE(ctx, value);
    return value;
Error:
    ctx->size = -1;  // Makes everything fail from now on.
    return -1;
}

//***ASN1NextTag()
// This function extracts the next type from 'buffer' starting at 'offset'.
// It advances 'offset' as it parses the type and the length of the type. It returns
// the length of the type. On return, the 'length' octets starting at 'offset' are the
```

```
// octets of the type.
// Return Type: UINT
//     >=0          the number of octets in 'type'
//     <0           an error
INT16
ASN1NextTag(ASN1UnmarshalContext* ctx)
{
    // A tag to get?
    GOTO_ERROR_UNLESS(ctx->offset < ctx->size);
    // Get it
    ctx->tag = NEXT_OCTET(ctx);
    // Make sure that it is not an extended tag
    GOTO_ERROR_UNLESS((ctx->tag & 0x1F) != 0x1F);
    // Get the length field and return that
    return ASN1DecodeLength(ctx);

Error:
    // Attempt to read beyond the end of the context or an illegal tag
    ctx->size = -1;  // Persistent failure
    ctx->tag  = 0xFF;
    return -1;
}

//*** ASN1GetBitStringValue()
// Try to parse a bit string of up to 32 bits from a value that is expected to be
// a bit string. The bit string is left justified so that the MSb of the input is
// the MSb of the returned value.
// If there is a general parsing error, the context->size is set to -1.
//  Return Type: BOOL
//      TRUE(1)     success
//      FALSE(0)    failure
BOOL ASN1GetBitStringValue(ASN1UnmarshalContext* ctx, UINT32* val)
{
    int    shift;
    INT16  length;
    UINT32 value = 0;
    int    inputBits;
    //
    length = ASN1NextTag(ctx);
    GOTO_ERROR_UNLESS(length >= 1);
    GOTO_ERROR_UNLESS(ctx->tag == ASN1_BITSTRING);
    // Get the shift value for the bit field (how many bits to lop off of the end)
    shift = NEXT_OCTET(ctx);
    length--;
    // Get the number of bits in the input
    inputBits = (8 * length) - shift;
    // the shift count has to make sense
    GOTO_ERROR_UNLESS((shift < 8) && ((length > 0) || (shift == 0)));
    // if there are any bytes left
    for(; length > 1; length--)
    {

        // for all but the last octet, just shift and add the new octet
        GOTO_ERROR_UNLESS((value & 0xFF000000) == 0);  // can't loose significant bits
        value = (value << 8) + NEXT_OCTET(ctx);
    }
    if(length == 1)
    {
        // for the last octet, just shift the accumulated value enough to
        // accept the significant bits in the last octet and shift the last
        // octet down
        GOTO_ERROR_UNLESS(((value & (0xFF000000 << (8 - shift)))) == 0);
        value = (value << (8 - shift)) + (NEXT_OCTET(ctx) >> shift);
    }
    // 'Left justify' the result
    if(inputBits > 0)
```

```
        value <<= (32 - inputBits);
    *val = value;
    return TRUE;
Error:
    ctx->size = -1;
    return FALSE;
}

//*****************************************************************
//** Marshaling Functions
//*****************************************************************

//*** Introduction
// Marshaling of an ASN.1 structure is accomplished from the bottom up. That is,
// the things that will be at the end of the structure are added last. To manage the
// collecting of the relative sizes, start a context for the outermost container, if
// there is one, and then placing items in from the bottom up. If the bottom-most
// item is also within a structure, create a nested context by calling
// ASN1StartMarshalingContext().
//
// The context control structure contains a 'buffer' pointer, an 'offset', an 'end'
// and a stack. 'offset' is the offset from the start of the buffer of the last added
// byte. When 'offset' reaches 0, the buffer is full. 'offset' is a signed value so
// that, when it becomes negative, there is an overflow. Only two functions are
// allowed to move bytes into the buffer: ASN1PushByte() and ASN1PushBytes(). These
// functions make sure that no data is written beyond the end of the buffer.
//
// When a new context is started, the current value of 'end' is pushed
// on the stack and 'end' is set to 'offset. As bytes are added, offset gets smaller.
// At any time, the count of bytes in the current context is simply 'end' - 'offset'.
//
// Since starting a new context involves setting 'end' = 'offset', the number of bytes
// in the context starts at 0. The nominal way of ending a context is to use
// 'end' - 'offset' to set the length value, and then a tag is added to the buffer.
// Then the previous 'end' value is popped meaning that the context just ended
// becomes a member of the now current context.
//
// The nominal strategy for building a completed ASN.1 structure is to push everything
// into the buffer and then move everything to the start of the buffer. The move is
// simple as the size of the move is the initial 'end' value minus the final 'offset'
// value. The destination is 'buffer' and the source is 'buffer' + 'offset'. As Skippy
// would say "Easy peasy, Joe."
//
// It is not necessary to provide a buffer into which the data is placed. If no buffer
// is provided, then the marshaling process will return values needed for marshaling.
// On strategy for filling the buffer would be to execute the process for building
// the structure without using a buffer. This would return the overall size of the
// structure. Then that amount of data could be allocated for the buffer and the fill
// process executed again with the data going into the buffer. At the end, the data
// would be in its final resting place.

//*** ASN1InitializeMarshalContext()
// This creates a structure for handling marshaling of an ASN.1 formatted data
// structure.
void ASN1InitializeMarshalContext(
    ASN1MarshalContext* ctx, INT16 length, BYTE* buffer)
{
    ctx->buffer = buffer;
    if(buffer)
        ctx->offset = length;
    else
        ctx->offset = INT16_MAX;
    ctx->end    = ctx->offset;
    ctx->depth = -1;
}
```

```
//*** ASN1StartMarshalContext()
// This starts a new constructed element. It is constructed on 'top' of the value
// that was previously placed in the structure.
void ASN1StartMarshalContext(ASN1MarshalContext* ctx)
{
    pAssert((ctx->depth + 1) < MAX_DEPTH);
    ctx->depth++;
    ctx->ends[ctx->depth] = ctx->end;
    ctx->end              = ctx->offset;
}

//*** ASN1EndMarshalContext()
// This function restores the end pointer for an encapsulating structure.
//  Return Type: INT16
//      > 0             the size of the encapsulated structure that was just ended
//      <= 0            an error
INT16
ASN1EndMarshalContext(ASN1MarshalContext* ctx)
{
    INT16 length;
    pAssert(ctx->depth >= 0);
    length   = ctx->end - ctx->offset;
    ctx->end = ctx->ends[ctx->depth--];
    return length;
}

//***ASN1EndEncapsulation()
// This function puts a tag and length in the buffer. In this function, an embedded
// BIT_STRING is assumed to be a collection of octets. To indicate that all bits
// are used, a byte of zero is prepended. If a raw bit-string is needed, a new
// function like ASN1PushInteger() would be needed.
//  Return Type: INT16
//      > 0         number of octets in the encapsulation
//      == 0        failure
UINT16
ASN1EndEncapsulation(ASN1MarshalContext* ctx, BYTE tag)
{
    // only add a leading zero for an encapsulated BIT STRING
    if(tag == ASN1_BITSTRING)
        ASN1PushByte(ctx, 0);
    ASN1PushTagAndLength(ctx, tag, ctx->end - ctx->offset);
    return ASN1EndMarshalContext(ctx);
}

//*** ASN1PushByte()
BOOL ASN1PushByte(ASN1MarshalContext* ctx, BYTE b)
{
    if(ctx->offset > 0)
    {
        ctx->offset -= 1;
        if(ctx->buffer)
            ctx->buffer[ctx->offset] = b;
        return TRUE;
    }
    ctx->offset = -1;
    return FALSE;
}

//*** ASN1PushBytes()
// Push some raw bytes onto the buffer. 'count' cannot be zero.
//  Return Type: IN16
//      > 0             count bytes
//      == 0            failure unless count was zero
INT16
ASN1PushBytes(ASN1MarshalContext* ctx, INT16 count, const BYTE* buffer)
{
```

```
    // make sure that count is not negative which would mess up the math; and that
    // if there is a count, there is a buffer
    GOTO_ERROR_UNLESS((count >= 0) && ((buffer != NULL) || (count == 0)));
    // back up the offset to determine where the new octets will get pushed
    ctx->offset -= count;
    // can't go negative
    GOTO_ERROR_UNLESS(ctx->offset >= 0);
    // if there are buffers, move the data, otherwise, assume that this is just a
    // test.
    if(count && buffer && ctx->buffer)
        MemoryCopy(&ctx->buffer[ctx->offset], buffer, count);
    return count;
Error:
    ctx->offset = -1;
    return 0;
}

//*** ASN1PushNull()
//  Return Type: IN16
//      > 0             count bytes
//      == 0            failure unless count was zero
INT16
ASN1PushNull(ASN1MarshalContext* ctx)
{
    ASN1PushByte(ctx, 0);
    ASN1PushByte(ctx, ASN1_NULL);
    return (ctx->offset >= 0) ? 2 : 0;
}

//*** ASN1PushLength()
// Push a length value. This will only handle length values that fit in an INT16.
//  Return Type: UINT16
//      > 0         number of bytes added
//      == 0        failure
INT16
ASN1PushLength(ASN1MarshalContext* ctx, INT16 len)
{
    UINT16 start = ctx->offset;
    GOTO_ERROR_UNLESS(len >= 0);
    if(len <= 127)
        ASN1PushByte(ctx, (BYTE)len);
    else
    {
        ASN1PushByte(ctx, (BYTE)(len & 0xFF));
        len >>= 8;
        if(len == 0)
            ASN1PushByte(ctx, 0x81);
        else
        {
            ASN1PushByte(ctx, (BYTE)(len));
            ASN1PushByte(ctx, 0x82);
        }
    }
    goto Exit;
Error:
    ctx->offset = -1;
Exit:
    return (ctx->offset > 0) ? start - ctx->offset : 0;
}

//*** ASN1PushTagAndLength()
//  Return Type: INT16
//      > 0         number of bytes added
//      == 0        failure
INT16
ASN1PushTagAndLength(ASN1MarshalContext* ctx, BYTE tag, INT16 length)
```

```
{
    INT16 bytes;
    bytes = ASN1PushLength(ctx, length);
    bytes += (INT16)ASN1PushByte(ctx, tag);
    return (ctx->offset < 0) ? 0 : bytes;
}

//*** ASN1PushTaggedOctetString()
// This function will push a random octet string.
//  Return Type: INT16
//      > 0         number of bytes added
//      == 0        failure
INT16
ASN1PushTaggedOctetString(
    ASN1MarshalContext* ctx, INT16 size, const BYTE* string, BYTE tag)
{
    ASN1PushBytes(ctx, size, string);
    // PushTagAndLenght just tells how many octets it added so the total size of this
    // element is the sum of those octets and input size.
    size += ASN1PushTagAndLength(ctx, tag, size);
    return size;
}

//*** ASN1PushUINT()
// This function pushes an native-endian integer value. This just changes a
// native-endian integer into a big-endian byte string and calls ASN1PushInteger().
// That function will remove leading zeros and make sure that the number is positive.
//  Return Type: IN16
//      > 0             count bytes
//      == 0            failure unless count was zero
INT16
ASN1PushUINT(ASN1MarshalContext* ctx, UINT32 integer)
{
    BYTE marshaled[4];
    UINT32_TO_BYTE_ARRAY(integer, marshaled);
    return ASN1PushInteger(ctx, 4, marshaled);
}

//*** ASN1PushInteger
// Push a big-endian integer on the end of the buffer
//  Return Type: UINT16
//      > 0         the number of bytes marshaled for the integer
//      == 0        failure
INT16
ASN1PushInteger(ASN1MarshalContext* ctx,     // IN/OUT: buffer context
                INT16               iLen,    // IN: octets of the integer
                BYTE*               integer  // IN: big-endian integer
)
{
    // no leading 0's
    while((*integer == 0) && (--iLen > 0))
        integer++;
    // Move the bytes to the buffer
    ASN1PushBytes(ctx, iLen, integer);
    // if needed, add a leading byte of 0 to make the number positive
    if(*integer & 0x80)
        iLen += (INT16)ASN1PushByte(ctx, 0);
    // PushTagAndLenght just tells how many octets it added so the total size of this
    // element is the sum of those octets and the adjusted input size.
    iLen += ASN1PushTagAndLength(ctx, ASN1_INTEGER, iLen);
    return iLen;
}

//*** ASN1PushOID()
// This function is used to add an OID. An OID is 0x06 followed by a byte of size
// followed by size bytes. This is used to avoid having to do anything special in the
```

```
// definition of an OID.
//  Return Type: UINT16
//      > 0         the number of bytes marshaled for the integer
//      == 0        failure
INT16
ASN1PushOID(ASN1MarshalContext* ctx, const BYTE* OID)
{
    if((*OID == ASN1_OBJECT_IDENTIFIER) && ((OID[1] & 0x80) == 0))
    {
        return ASN1PushBytes(ctx, OID[1] + 2, OID);
    }
    ctx->offset = -1;
    return 0;
}

#endif  // CC_CertifyX509
```

7.201 /tpm/src/X509/X509_ECC.c

```
//** Includes
#include "Tpm.h"
#include "X509.h"
#include "OIDs.h"
#include "TpmASN1_fp.h"
#include "X509_ECC_fp.h"
#include "X509_spt_fp.h"
#include "CryptHash_fp.h"

//** Functions

//*** X509PushPoint()
// This seems like it might be used more than once so...
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509PushPoint(ASN1MarshalContext* ctx, TPMS_ECC_POINT* p)
{
    // Push a bit string containing the public key. For now, push the x, and y
    // coordinates of the public point, bottom up
    ASN1StartMarshalContext(ctx);  // BIT STRING
    {
        ASN1PushBytes(ctx, p->y.t.size, p->y.t.buffer);
        ASN1PushBytes(ctx, p->x.t.size, p->x.t.buffer);
        ASN1PushByte(ctx, 0x04);
    }
    return ASN1EndEncapsulation(ctx, ASN1_BITSTRING);  // Ends BIT STRING
}

//*** X509AddSigningAlgorithmECC()
// This creates the singing algorithm data.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddSigningAlgorithmECC(
    OBJECT* signKey, TPMT_SIG_SCHEME* scheme, ASN1MarshalContext* ctx)
{
    PHASH_DEF hashDef = CryptGetHashDef(scheme->details.any.hashAlg);
    //
    NOT_REFERENCED(signKey);
    // If the desired hashAlg definition wasn't found...
    if(hashDef->hashAlg != scheme->details.any.hashAlg)
        return 0;
```

```
    switch(scheme->scheme)
    {
#if ALG_ECDSA
        case TPM_ALG_ECDSA:
            // Make sure that we have an OID for this hash and ECC
            if((hashDef->ECDSA)[0] != ASN1_OBJECT_IDENTIFIER)
                break;
            // if this is just an implementation check, indicate that this
            // combination is supported
            if(!ctx)
                return 1;
            ASN1StartMarshalContext(ctx);
            ASN1PushOID(ctx, hashDef->ECDSA);
            return ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);
#endif  //  ALG_ECDSA
        default:
            break;
    }
    return 0;
}

//*** X509AddPublicECC()
// This function will add the publicKey description to the DER data. If ctx is
// NULL, then no data is transferred and this function will indicate if the TPM
// has the values for DER-encoding of the public key.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddPublicECC(OBJECT* object, ASN1MarshalContext* ctx)
{
    const BYTE* curveOid =
        CryptEccGetOID(object->publicArea.parameters.eccDetail.curveID);
    if((curveOid == NULL) || (*curveOid != ASN1_OBJECT_IDENTIFIER))
        return 0;
    //
    //
    //  SEQUENCE (2 elem) 1st
    //    SEQUENCE (2 elem) 2nd
    //      OBJECT IDENTIFIER 1.2.840.10045.2.1 ecPublicKey (ANSI X9.62 public key
type)
    //      OBJECT IDENTIFIER 1.2.840.10045.3.1.7 prime256v1 (ANSI X9.62 named curve)
    //    BIT STRING (520 bit)
000001001010000111010101010111001001101101000100000010...
    //
    // If this is a check to see if the key can be encoded, it can.
    // Need to mark the end sequence
    if(ctx == NULL)
        return 1;
    ASN1StartMarshalContext(ctx);  // SEQUENCE (2 elem) 1st
    {
        X509PushPoint(ctx, &object->publicArea.unique.ecc);  // BIT STRING
        ASN1StartMarshalContext(ctx);                        // SEQUENCE (2 elem) 2nd
        {
            ASN1PushOID(ctx, curveOid);         // curve dependent
            ASN1PushOID(ctx, OID_ECC_PUBLIC);   // (1.2.840.10045.2.1)
        }
        ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);  // Ends SEQUENCE 2nd
    }
    return ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);  // Ends SEQUENCE 1st
}
```

7.202 /tpm/src/X509/X509_RSA.c

```
//** Includes
```

```
#include "Tpm.h"
#include "X509.h"
#include "TpmASN1_fp.h"
#include "X509_RSA_fp.h"
#include "X509_spt_fp.h"
#include "CryptHash_fp.h"
#include "CryptRsa_fp.h"

//** Functions

#if ALG_RSA

//*** X509AddSigningAlgorithmRSA()
// This creates the singing algorithm data.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddSigningAlgorithmRSA(
    OBJECT* signKey, TPMT_SIG_SCHEME* scheme, ASN1MarshalContext* ctx)
{
    TPM_ALG_ID hashAlg = scheme->details.any.hashAlg;
    PHASH_DEF  hashDef = CryptGetHashDef(hashAlg);
    //
    NOT_REFERENCED(signKey);
    // return failure if hash isn't implemented
    if(hashDef->hashAlg != hashAlg)
        return 0;
    switch(scheme->scheme)
    {
        case TPM_ALG_RSASSA:
        {
            // if the hash is implemented but there is no PKCS1 OID defined
            // then this is not a valid signing combination.
            if(hashDef->PKCS1[0] != ASN1_OBJECT_IDENTIFIER)
                break;
            if(ctx == NULL)
                return 1;
            return X509PushAlgorithmIdentifierSequence(ctx, hashDef->PKCS1);
        }
        case TPM_ALG_RSAPSS:
            // leave if this is just an implementation check
            if(ctx == NULL)
                return 1;
            // In the case of SHA1, everything is default and RFC4055 says that
            // implementations that do signature generation MUST omit the parameter
            // when defaults are used. )-:
            if(hashDef->hashAlg == TPM_ALG_SHA1)
            {
                return X509PushAlgorithmIdentifierSequence(ctx, OID_RSAPSS);
            }
            else
            {
                // Going to build something that looks like:
                //  SEQUENCE (2 elem)
                //     OBJECT IDENTIFIER 1.2.840.113549.1.1.10 rsaPSS (PKCS #1)
                //     SEQUENCE (3 elem)
                //       [0] (1 elem)
                //         SEQUENCE (2 elem)
                //           OBJECT IDENTIFIER 2.16.840.1.101.3.4.2.1 sha-256
                //           NULL
                //       [1] (1 elem)
                //         SEQUENCE (2 elem)
                //           OBJECT IDENTIFIER 1.2.840.113549.1.1.8 pkcs1-MGF
                //           SEQUENCE (2 elem)
                //             OBJECT IDENTIFIER 2.16.840.1.101.3.4.2.1 sha-256
```

```
//             NULL
//       [2] (1 elem)  salt length
//         INTEGER 32

// The indentation is just to keep track of where we are in the
// structure
ASN1StartMarshalContext(ctx);  // SEQUENCE (2 elements)
{
    ASN1StartMarshalContext(ctx);  // SEQUENCE (3 elements)
    {
        // [2] (1 elem)  salt length
        //    INTEGER 32
        ASN1StartMarshalContext(ctx);
        {
            INT16 saltSize = CryptRsaPssSaltSize(
                (INT16)hashDef->digestSize,
                (INT16)signKey->publicArea.unique.rsa.t.size);
            ASN1PushUINT(ctx, saltSize);
        }
        ASN1EndEncapsulation(ctx, ASN1_APPLICAIION_SPECIFIC + 2);

        // Add the mask generation algorithm
        // [1] (1 elem)
        //    SEQUENCE (2 elem) 1st
        //      OBJECT IDENTIFIER 1.2.840.113549.1.1.8 pkcs1-MGF
        //      SEQUENCE (2 elem) 2nd
        //        OBJECT IDENTIFIER 2.16.840.1.101.3.4.2.1 sha-256
        //        NULL
        ASN1StartMarshalContext(ctx);  // mask context [1] (1 elem)
        {
            ASN1StartMarshalContext(ctx);  // SEQUENCE (2 elem) 1st
            // Handle the 2nd Sequence (sequence (object, null))
            {
                // This adds a NULL, then an OID and a SEQUENCE
                // wrapper.
                X509PushAlgorithmIdentifierSequence(ctx,
                                                    hashDef->OID);
                // add the pkcs1-MGF OID
                ASN1PushOID(ctx, OID_MGF1);
            }
            // End outer sequence
            ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);
        }
        // End the [1]
        ASN1EndEncapsulation(ctx, ASN1_APPLICAIION_SPECIFIC + 1);

        // Add the hash algorithm
        // [0] (1 elem)
        //   SEQUENCE (2 elem) (done by
        //               X509PushAlgorithmIdentifierSequence)
        //     OBJECT IDENTIFIER 2.16.840.1.101.3.4.2.1 sha-256 (NIST)
        //     NULL
        ASN1StartMarshalContext(ctx);  // [0] (1 elem)
        {
            X509PushAlgorithmIdentifierSequence(ctx, hashDef->OID);
        }
        ASN1EndEncapsulation(ctx, (ASN1_APPLICAIION_SPECIFIC + 0));
    }
    //  SEQUENCE (3 elements) end
    ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);

    // RSA PSS OID
    // OBJECT IDENTIFIER 1.2.840.113549.1.1.10 rsaPSS (PKCS #1)
    ASN1PushOID(ctx, OID_RSAPSS);
}
// End Sequence (2 elements)
```

```
                return ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);
            }
        default:
            break;
    }
    return 0;
}

//*** X509AddPublicRSA()
// This function will add the publicKey description to the DER data. If fillPtr is
// NULL, then no data is transferred and this function will indicate if the TPM
// has the values for DER-encoding of the public key.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509AddPublicRSA(OBJECT* object, ASN1MarshalContext* ctx)
{
    UINT32 exp = object->publicArea.parameters.rsaDetail.exponent;
    //
    /*
    SEQUENCE (2 elem) 1st
      SEQUENCE (2 elem) 2nd
        OBJECT IDENTIFIER 1.2.840.113549.1.1.1 rsaEncryption (PKCS #1)
        NULL
      BIT STRING (1 elem)
        SEQUENCE (2 elem) 3rd
          INTEGER (2048 bit) 21973045137412279557258341993574O1
          INTEGER 65537
*/
    // If this is a check to see if the key can be encoded, it can.
    // Need to mark the end sequence
    if(ctx == NULL)
        return 1;
    ASN1StartMarshalContext(ctx);  // SEQUENCE (2 elem) 1st
    ASN1StartMarshalContext(ctx);  // BIT STRING
    ASN1StartMarshalContext(ctx);  // SEQUENCE *(2 elem) 3rd

    // Get public exponent in big-endian byte order.
    if(exp == 0)
        exp = RSA_DEFAULT_PUBLIC_EXPONENT;

    // Push a 4 byte integer. This might get reduced if there are leading zeros or
    // extended if the high order byte is negative.
    ASN1PushUINT(ctx, exp);
    // Push the public key as an integer
    ASN1PushInteger(ctx,
                    object->publicArea.unique.rsa.t.size,
                    object->publicArea.unique.rsa.t.buffer);
    // Embed this in a SEQUENCE tag and length in for the key, exponent sequence
    ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);  // SEQUENCE (3rd)

    // Embed this in a BIT STRING
    ASN1EndEncapsulation(ctx, ASN1_BITSTRING);

    // Now add the formatted SEQUENCE for the RSA public key OID. This is a
    // fully constructed value so it doesn't need to have a context started
    X509PushAlgorithmIdentifierSequence(ctx, OID_PKCS1_PUB);

    return ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);
}

#endif  // ALG_RSA
```

7.203 /tpm/src/X509/X509_spt.c

```
//** Includes
#include "Tpm.h"
#include "TpmASN1.h"
#include "TpmASN1_fp.h"
#define _X509_SPT_
#include "X509.h"
#include "X509_spt_fp.h"
#if ALG_RSA
#  include "X509_RSA_fp.h"
#endif  // ALG_RSA
#if ALG_ECC
#  include "X509_ECC_fp.h"
#endif  // ALG_ECC
#if ALG_SM2
//#   include "X509_SM2_fp.h"
#endif  // ALG_RSA

#if CC_CertifyX509

//** Unmarshaling Functions

//*** X509FindExtensionByOID()
// This will search a list of X509 extensions to find an extension with the
// requested OID. If the extension is found, the output context ('ctx') is set up
// to point to the OID in the extension.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure (could be catastrophic)
BOOL X509FindExtensionByOID(ASN1UnmarshalContext* ctxIn,  // IN: the context to search
                            ASN1UnmarshalContext* ctx,  // OUT: the extension context
                            const BYTE*           OID   // IN: oid to search for
)
{
    INT16 length;
    //
    pAssert(ctxIn != NULL);
    // Make the search non-destructive of the input if ctx provided. Otherwise, use
    // the provided context.
    if(ctx == NULL)
        ctx = ctxIn;
    // if the provided search context is different from the context of the extension,
    // then copy the search context to the search context.
    else if(ctx != ctxIn)
        *ctx = *ctxIn;
    // Now, search in the extension context
    for(; ctx->size > ctx->offset; ctx->offset += length)
    {
        GOTO_ERROR_UNLESS((length = ASN1NextTag(ctx)) >= 0);
        // If this is not a constructed sequence, then it doesn't belong
        // in the extensions.
        GOTO_ERROR_UNLESS(ctx->tag == ASN1_CONSTRUCTED_SEQUENCE);
        // Make sure that this entry could hold the OID
        if(length >= OID_SIZE(OID))
        {
            // See if this is a match for the provided object identifier.
            if(MemoryEqual(OID, &(ctx->buffer[ctx->offset]), OID_SIZE(OID)))
            {
                // Return with ' ctx' set to point to the start of the OID with the
size
                // set to be the size of the SEQUENCE
                ctx->buffer += ctx->offset;
                ctx->offset = 0;
                ctx->size   = length;
                return TRUE;
```

```
            }
        }
    }
    GOTO_ERROR_UNLESS(ctx->offset == ctx->size);
    return FALSE;
Error:
    ctxIn->size = -1;
    ctx->size   = -1;
    return FALSE;
}

//*** X509GetExtensionBits()
// This function will extract a bit field from an extension. If the extension doesn't
// contain a bit string, it will fail.
// Return Type: BOOL
//  TRUE(1)         success
//  FALSE(0)        failure
UINT32
X509GetExtensionBits(ASN1UnmarshalContext* ctx, UINT32* value)
{
    INT16 length;
    //
    while(((length = ASN1NextTag(ctx)) > 0) && (ctx->size > ctx->offset))
    {
        // Since this is an extension, the extension value will be in an OCTET STRING
        if(ctx->tag == ASN1_OCTET_STRING)
        {
            return ASN1GetBitStringValue(ctx, value);
        }
        ctx->offset += length;
    }
    ctx->size = -1;
    return FALSE;
}

//***X509ProcessExtensions()
// This function is used to process the TPMA_OBJECT and KeyUsage extensions. It is not
// in the CertifyX509.c code because it makes the code harder to follow.
// Return Type: TPM_RC
//      TPM_RCS_ATTRIBUTES      the attributes of object are not consistent with
//                              the extension setting
//      TPM_RC_VALUE            problem parsing the extensions
TPM_RC
X509ProcessExtensions(
    OBJECT* object,         // IN: The object with the attributes to
                            //      check
    stringRef* extension    // IN: The start and length of the extensions
)
{
    ASN1UnmarshalContext ctx;
    ASN1UnmarshalContext extensionCtx;
    INT16                length;
    UINT32               value;
    TPMA_OBJECT          attributes = object->publicArea.objectAttributes;
    //
    if(!ASN1UnmarshalContextInitialize(&ctx, extension->len, extension->buf)
       || ((length = ASN1NextTag(&ctx)) < 0) || (ctx.tag != X509_EXTENSIONS))
        return TPM_RCS_VALUE;
    if(((length = ASN1NextTag(&ctx)) < 0) || (ctx.tag != (ASN1_CONSTRUCTED_SEQUENCE)))
        return TPM_RCS_VALUE;

    // Get the extension for the TPMA_OBJECT if there is one
    if(X509FindExtensionByOID(&ctx, &extensionCtx, OID_TCG_TPMA_OBJECT)
       && X509GetExtensionBits(&extensionCtx, &value))
    {
        // If an keyAttributes extension was found, it must be exactly the same as the
```

```
        // attributes of the object.
        // NOTE: MemoryEqual() is used rather than a simple UINT32 compare to avoid
        // type-punned pointer warning/error.
        if(!MemoryEqual(&value, &attributes, sizeof(value)))
            return TPM_RCS_ATTRIBUTES;
    }
    // Make sure the failure to find the value wasn't because of a fatal error
    else if(extensionCtx.size < 0)
        return TPM_RCS_VALUE;

    // Get the keyUsage extension. This one is required
    if(X509FindExtensionByOID(&ctx, &extensionCtx, OID_KEY_USAGE_EXTENSION)
       && X509GetExtensionBits(&extensionCtx, &value))
    {
        x509KeyUsageUnion keyUsage;
        BOOL              badSign;
        BOOL              badDecrypt;
        BOOL              badFixedTPM;
        BOOL              badRestricted;

        //
        keyUsage.integer = value;

        // see if any reserved bits are set
        if(keyUsage.integer & ~(TPMA_X509_KEY_USAGE_ALLOWED_BITS))
            return TPM_RCS_RESERVED_BITS;

        // For KeyUsage:
        // 1) 'sign' is SET if Key Usage includes signing
        badSign = ((KEY_USAGE_SIGN.integer & keyUsage.integer) != 0)
                  && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign);
        // 2) 'decrypt' is SET if Key Usage includes decryption uses
        badDecrypt = ((KEY_USAGE_DECRYPT.integer & keyUsage.integer) != 0)
                     && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt);
        // 3) 'fixedTPM' is SET if Key Usage is non-repudiation
        badFixedTPM = IS_ATTRIBUTE(keyUsage.x509, TPMA_X509_KEY_USAGE, nonrepudiation)
                      && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM);
        // 4)'restricted' is SET if Key Usage is for key encipherment.
        badRestricted =
            IS_ATTRIBUTE(keyUsage.x509, TPMA_X509_KEY_USAGE, keyEncipherment)
            && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted);
        if(badSign || badDecrypt || badFixedTPM || badRestricted)
            return TPM_RCS_VALUE;
    }
    else
        // The KeyUsage extension is required
        return TPM_RCS_VALUE;

    return TPM_RC_SUCCESS;
}

//** Marshaling Functions

//*** X509AddSigningAlgorithm()
// This creates the singing algorithm data.
// Return Type: INT16
//  > 0                 number of octets added
// <= 0                 failure
INT16
X509AddSigningAlgorithm(
    ASN1MarshalContext* ctx, OBJECT* signKey, TPMT_SIG_SCHEME* scheme)
{
    switch(signKey->publicArea.type)
    {
#  if ALG_RSA
        case TPM_ALG_RSA:
```

```
            return X509AddSigningAlgorithmRSA(signKey, scheme, ctx);
#   endif  // ALG_RSA
#   if ALG_ECC
        case TPM_ALG_ECC:
            return X509AddSigningAlgorithmECC(signKey, scheme, ctx);
#   endif  // ALG_ECC
#   if ALG_SM2
        case TPM_ALG_SM2:
            break;  // no signing algorithm for SM2 yet
//            return X509AddSigningAlgorithmSM2(signKey, scheme, ctx);
#   endif  // ALG_SM2
        default:
            break;
    }
    return 0;
}

//*** X509AddPublicKey()
// This function will add the publicKey description to the DER data. If fillPtr is
// NULL, then no data is transferred and this function will indicate if the TPM
// has the values for DER-encoding of the public key.
//  Return Type: INT16
//      > 0         number of octets added
//      == 0        failure
INT16
X509AddPublicKey(ASN1MarshalContext* ctx, OBJECT* object)
{
    switch(object->publicArea.type)
    {
#   if ALG_RSA
        case TPM_ALG_RSA:
            return X509AddPublicRSA(object, ctx);
#   endif
#   if ALG_ECC
        case TPM_ALG_ECC:
            return X509AddPublicECC(object, ctx);
#   endif
#   if ALG_SM2
        case TPM_ALG_SM2:
            break;
#   endif
        default:
            break;
    }
    return FALSE;
}

//*** X509PushAlgorithmIdentifierSequence()
// The function adds the algorithm identifier sequence.
//  Return Type: INT16
//      > 0         number of bytes added
//     == 0         failure
INT16
X509PushAlgorithmIdentifierSequence(ASN1MarshalContext* ctx, const BYTE* OID)
{
    // An algorithm ID sequence is:
    //  SEQUENCE
    //      OID
    //      NULL
    ASN1StartMarshalContext(ctx);  // hash algorithm
    ASN1PushNull(ctx);
    ASN1PushOID(ctx, OID);
    return ASN1EndEncapsulation(ctx, ASN1_CONSTRUCTED_SEQUENCE);
}

#endif  // CC_CertifyX509
```

Annex A
(informative)
Implementation Dependent

A.1 Introduction

These files contains definitions that are used to define a TPM profile. The values are chosen by the manufacturer. The values here are chosen to represent a full featured TPM so that all of the TPM's capabilities can be simulated and tested. This file would change based on the implementation.

The file listed below was generated by an automated tool using three documents as inputs. They are:

1) The TCG Algorithm Registry,
2) Part 2 of this specification, and
3) A purpose-built document that contains vendor-specific information in tables.
4) All of the values in this file have #ifdef *'guards'* so that they may be defined in a command line. Additionally, TpmBuildSwitches.h allows an additional file to be specified in the compiler command line and preset any of these values.

/TpmConfiguration/TpmConfiguration/TpmBuildSwitches.h

```
// This file contains the build switches. This contains switches for multiple
// versions of the crypto-library so some may not apply to your environment.
// Each switch has an accompanying description below.
//
// clang-format off
#ifndef _TPM_BUILD_SWITCHES_H_
#define _TPM_BUILD_SWITCHES_H_

#if defined(YES) || defined(NO)
#  error YES and NO should be defined in TpmBuildSwitches.h
#endif
#if defined(SET) || defined(CLEAR)
#  error SET and CLEAR should be defined in TpmBuildSwitches.h
#endif

#define YES   1
#define SET   1
#define NO    0
#define CLEAR 0

// TRUE/FALSE may be coming from system headers, but if not, provide them.
#ifndef TRUE
#  define TRUE 1
#endif
#ifndef FALSE
#  define FALSE 0
#endif

// Need an unambiguous definition for DEBUG. Do not change this
#ifndef DEBUG
#  ifdef NDEBUG
#    define DEBUG NO
#  else
#    define DEBUG YES
#  endif
#elif(DEBUG != NO) && (DEBUG != YES)
#  error DEBUG should be 0 or 1
#endif
```

```
//////////////////////////////////////////////////////////////
// DEBUG OPTIONS
//////////////////////////////////////////////////////////////

// The SIMULATION switch allows certain other macros to be enabled. The things that
// can be enabled in a simulation include key caching, reproducible "random"
// sequences, instrumentation of the RSA key generation process, and certain other
// debug code. SIMULATION Needs to be defined as either YES or NO. This grouping of
// macros will make sure that it is set correctly. A simulated TPM would include a
// Virtual TPM. The interfaces for a Virtual TPM should be modified from the standard
// ones in the Simulator project.
#define SIMULATION                  YES


// The CRYPTO_LIB_REPORTING switch allows the TPM to report its
// crypto library implementation, e.g., at simulation startup.
#define CRYPTO_LIB_REPORTING        YES

// If doing debug, can set the DRBG to print out the intermediate test values.
// Before enabling this, make sure that the dbgDumpMemBlock() function
// has been added someplace (preferably, somewhere in CryptRand.c)
#define DRBG_DEBUG_PRINT            (NO  * DEBUG)

// This define is used to control the debug for the CertifyX509 command.
#define CERTIFYX509_DEBUG           (YES * DEBUG)

// This provides fixed seeding of the RNG when doing debug on a simulator. This
// should allow consistent results on test runs as long as the input parameters
// to the functions remains the same.
#define USE_DEBUG_RNG               (NO  * DEBUG)

//////////////////////////////////////////////////////////////
// RSA DEBUG OPTIONS
//////////////////////////////////////////////////////////////

// Enable the instrumentation of the sieve process. This is used to tune the sieve
// variables.
#define RSA_INSTRUMENT              (NO  * DEBUG)

// Enables use of the key cache. Default is YES
#define USE_RSA_KEY_CACHE           (NO  * DEBUG)

// Enables use of a file to store the key cache values so that the TPM will start
// faster during debug. Default for this is YES
#define USE_KEY_CACHE_FILE          (NO  * DEBUG)

//////////////////////////////////////////////////////////////
// TEST OPTIONS
//////////////////////////////////////////////////////////////
// The SIMULATION flag can enable test crypto behaviors and caching that
// significantly change the behavior of the code.  This flag controls only the
// g_forceFailureMode flag in the TPM library while leaving the rest of the TPM
// behavior alone.  Useful for testing when the full set of options controlled by
// SIMULATION may not be desired.
#define ALLOW_FORCE_FAILURE_MODE    YES

//////////////////////////////////////////////////////////////
// Internal checks
//////////////////////////////////////////////////////////////

// Define this to run the function that checks the compatibility between the
// chosen big number math library and the TPM code. Not all ports use this.
#define LIBRARY_COMPATIBILITY_CHECK YES

// In some cases, the relationship between two values may be dependent on things that
// change based on various selections like the chosen cryptographic libraries. It is
```

```
// possible that these selections will result in incompatible settings. These are
often
// detectable by the compiler but it is not always possible to do the check in the
// preprocessor code. For example, when the check requires use of 'sizeof'() then the
// preprocessor can't do the comparison. For these cases, we include a special macro
// that, depending on the compiler will generate a warning to indicate if the check
// always passes or always fails because it involves fixed constants.
//
// In modern compilers this is now commonly known as a static_assert, but the precise
// implementation varies by compiler. CompilerDependencies.h defines MUST_BE as a
macro
// that abstracts out the differences, and COMPILER_CHECKS can remove the checks where
// the current compiler doesn't support it.  COMPILER_CHECKS should be enabled if the
// compiler supports some form of static_assert.
// See the CompilerDependencies_*.h files for specific implementations per compiler.
#define COMPILER_CHECKS             YES

// Some of the values (such as sizes) are the result of different options set in
// TpmProfile.h. The combination might not be consistent. A function is defined
// (TpmSizeChecks()) that is used to verify the sizes at run time. To enable the
// function, define this parameter.
#define RUNTIME_SIZE_CHECKS         YES

//////////////////////////////////////////////////////////////////
// Compliance options
//////////////////////////////////////////////////////////////////

// Enable extra behaviors to meet FIPS compliance requirements
#define FIPS_COMPLIANT              YES

// Indicates if the implementation is to compute the sizes of the proof and primary
// seed size values based on the implemented algorithms.
#define USE_SPEC_COMPLIANT_PROOFS   YES

// Set this to allow compile to continue even though the chosen proof values
// do not match the compliant values. This is written so that someone would
// have to proactively ignore errors.
#define SKIP_PROOF_ERRORS           NO

//////////////////////////////////////////////////////////////////
// Implementation alternatives - don't  change external behavior
//////////////////////////////////////////////////////////////////

// Define TABLE_DRIVEN_DISPATCH to use tables rather than case statements
// for command dispatch and handle unmarshaling
#define TABLE_DRIVEN_DISPATCH       YES

// This define is used to enable the new table-driven marshaling code.
#define TABLE_DRIVEN_MARSHAL        NO

// This switch allows use of #defines in place of pass-through marshaling or
// unmarshaling code. A pass-through function just calls another function to do
// the required function and does no parameter checking of its own. The
// table-driven dispatcher calls directly to the lowest level
// marshaling/unmarshaling code and by-passes any pass-through functions.
#define USE_MARSHALING_DEFINES      YES

// Switch added to support packed lists that leave out space associated with
// unimplemented commands. Comment this out to use linear lists.
// Note: if vendor specific commands are present, the associated list is always
// in compressed form.
#define COMPRESSED_LISTS            YES

// This define is used to eliminate the use of bit-fields. It can be enabled for big-
// or little-endian machines. For big-endian architectures that numbers bits in
// registers from left to right (MSb0) this must be enabled. Little-endian machines
```

```
// number from right to left with the least significant bit having assigned a bit
// number of 0. These are LSb0 machines (they are also little-endian so they are also
// least-significant byte 0 (LSB0) machines. Big-endian (MSB0) machines may number in
// either direction (MSb0 or LSb0). For an MSB0+MSb0 machine this value is required to
// be 'NO'
#define USE_BIT_FIELD_STRUCTURES    NO

// Enable the generation of RSA primes using a sieve.
#define RSA_KEY_SIEVE               YES

////////////////////////////////////////////////////////////////
// Implementation alternatives - changes external behavior
////////////////////////////////////////////////////////////////

// This switch enables the RNG state save and restore
#define _DRBG_STATE_SAVE            YES

// Definition to allow alternate behavior for non-orderly startup. If there is a
// chance that the TPM could not update 'failedTries'
#define USE_DA_USED                 YES

// This switch is used to enable the self-test capability in AlgorithmTests.c
#define ENABLE_SELF_TESTS           YES

// This switch indicates where clock epoch value should be stored. If this value
// defined, then it is assumed that the timer will change at any time so the
// nonce should be a random number kept in RAM. When it is not defined, then the
// timer only stops during power outages.
#define CLOCK_STOPS                 NO

// Indicate if the implementation is going to give lockout time credit for time up to
// the last orderly shutdown.
#define ACCUMULATE_SELF_HEAL_TIMER  YES

// If an assertion event is not going to produce any trace information (function and
// line number) then make FAIL_TRACE == NO
#define FAIL_TRACE                  YES

// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/CompilerDependencies.h>

#endif  // _TPM_BUILD_SWITCHES_H_
```

/TpmConfiguration/TpmConfiguration/TpmProfile.h

```
// The primary configuration file that collects all configuration options for a
// TPM build.
#ifndef _TPM_PROFILE_H_
#define _TPM_PROFILE_H_

#include <TpmConfiguration/TpmBuildSwitches.h>
#include <TpmConfiguration/TpmProfile_Common.h>
#include <TpmConfiguration/TpmProfile_CommandList.h>
#include <TpmConfiguration/TpmProfile_Misc.h>
#include <TpmConfiguration/TpmProfile_ErrorCodes.h>
#include <TpmConfiguration/VendorInfo.h>

#endif  // _TPM_PROFILE_H_
```

/TpmConfiguration/TpmConfiguration/TpmProfile_CommandList.h

```
// this file defines the desired command list that should be built into the
// Tpm Core Lib.
```

```
#ifndef _TPM_PROFILE_COMMAND_LIST_H_
#define _TPM_PROFILE_COMMAND_LIST_H_

#if(YES != 1 || NO != 0)
#  error YES and NO must be correctly set before including TpmProfile_CommandList.h
#endif
#if defined(CC_YES) || defined(CC_NO)
#  error CC_YES and CC_NO should be defined by the command line file, not before
#endif

#define CC_YES YES
#define CC_NO  NO

//
// Defines for Implemented Commands
//

// Commands that are defined in the spec, but not implemented for various
// reasons:

// The TPM reference implementation does not implement attached-component
// features, and the Compliance test suite has no test cases.
#define CC_AC_GetCapability CC_NO
#define CC_AC_Send          CC_NO

// The TPM reference implementation does not implement firmware upgrade.
#define CC_FieldUpgradeData  CC_NO
#define CC_FieldUpgradeStart CC_NO
#define CC_FirmwareRead      CC_NO

// A prototype of CertifyX509 is provided here for informative purposes only.
// While all of the TPM reference implementation is provided "AS IS" without any
// warranty, the current design and implementation of CertifyX509 are considered
// to be especially unsuitable for product use.
#define CC_CertifyX509 CC_NO

// Normal commands:

#define CC_ACT_SetTimeout             (CC_YES && ACT_SUPPORT)
#define CC_ActivateCredential         CC_YES
#define CC_Certify                    CC_YES
#define CC_CertifyCreation            CC_YES
#define CC_ChangeEPS                  CC_YES
#define CC_ChangePPS                  CC_YES
#define CC_Clear                      CC_YES
#define CC_ClearControl               CC_YES
#define CC_ClockRateAdjust            CC_YES
#define CC_ClockSet                   CC_YES
#define CC_Commit                     (CC_YES && ALG_ECC)
#define CC_ContextLoad                CC_YES
#define CC_ContextSave                CC_YES
#define CC_Create                     CC_YES
#define CC_CreateLoaded               CC_YES
#define CC_CreatePrimary              CC_YES
#define CC_DictionaryAttackLockReset  CC_YES
#define CC_DictionaryAttackParameters CC_YES
#define CC_Duplicate                  CC_YES
#define CC_ECC_Decrypt                (CC_YES && ALG_ECC)
#define CC_ECC_Encrypt                (CC_YES && ALG_ECC)
#define CC_ECC_Parameters             (CC_YES && ALG_ECC)
#define CC_ECDH_KeyGen                (CC_YES && ALG_ECC)
#define CC_ECDH_ZGen                  (CC_YES && ALG_ECC)
#define CC_EC_Ephemeral               (CC_YES && ALG_ECC)
#define CC_EncryptDecrypt             CC_YES
#define CC_EncryptDecrypt2            CC_YES
#define CC_EventSequenceComplete      CC_YES
```

```
#define CC_EvictControl              CC_YES
#define CC_FlushContext              CC_YES
#define CC_GetCapability             CC_YES
#define CC_GetCommandAuditDigest     CC_YES
#define CC_GetRandom                 CC_YES
#define CC_GetSessionAuditDigest     CC_YES
#define CC_GetTestResult             CC_YES
#define CC_GetTime                   CC_YES
#define CC_HMAC                      (CC_YES && !ALG_CMAC)
#define CC_HMAC_Start                (CC_YES && !ALG_CMAC)
#define CC_Hash                      CC_YES
#define CC_HashSequenceStart         CC_YES
#define CC_HierarchyChangeAuth       CC_YES
#define CC_HierarchyControl          CC_YES
#define CC_Import                    CC_YES
#define CC_IncrementalSelfTest       CC_YES
#define CC_Load                      CC_YES
#define CC_LoadExternal              CC_YES
#define CC_MAC                       (CC_YES && ALG_CMAC)
#define CC_MAC_Start                 (CC_YES && ALG_CMAC)
#define CC_MakeCredential            CC_YES
#define CC_NV_Certify                CC_YES
#define CC_NV_ChangeAuth             CC_YES
#define CC_NV_DefineSpace            CC_YES
#define CC_NV_Extend                 CC_YES
#define CC_NV_GlobalWriteLock        CC_YES
#define CC_NV_Increment              CC_YES
#define CC_NV_Read                   CC_YES
#define CC_NV_ReadLock               CC_YES
#define CC_NV_ReadPublic             CC_YES
#define CC_NV_SetBits                CC_YES
#define CC_NV_UndefineSpace          CC_YES
#define CC_NV_UndefineSpaceSpecial   CC_YES
#define CC_NV_Write                  CC_YES
#define CC_NV_WriteLock              CC_YES
#define CC_ObjectChangeAuth          CC_YES
#define CC_PCR_Allocate              CC_YES
#define CC_PCR_Event                 CC_YES
#define CC_PCR_Extend                CC_YES
#define CC_PCR_Read                  CC_YES
#define CC_PCR_Reset                 CC_YES
#define CC_PCR_SetAuthPolicy         CC_YES
#define CC_PCR_SetAuthValue          CC_YES
#define CC_PP_Commands               CC_YES
#define CC_PolicyAuthValue           CC_YES
#define CC_PolicyAuthorize           CC_YES
#define CC_PolicyAuthorizeNV         CC_YES
#define CC_PolicyCapability          CC_YES
#define CC_PolicyCommandCode         CC_YES
#define CC_PolicyCounterTimer        CC_YES
#define CC_PolicyCpHash              CC_YES
#define CC_PolicyDuplicationSelect   CC_YES
#define CC_PolicyGetDigest           CC_YES
#define CC_PolicyLocality            CC_YES
#define CC_PolicyNV                  CC_YES
#define CC_PolicyNameHash            CC_YES
#define CC_PolicyNvWritten           CC_YES
#define CC_PolicyOR                  CC_YES
#define CC_PolicyPCR                 CC_YES
#define CC_PolicyPassword            CC_YES
#define CC_PolicyParameters          CC_YES
#define CC_PolicyPhysicalPresence    CC_YES
#define CC_PolicyRestart             CC_YES
#define CC_PolicySecret              CC_YES
#define CC_PolicySigned              CC_YES
#define CC_PolicyTemplate            CC_YES
```

```
#define CC_PolicyTicket                 CC_YES
#define CC_Policy_AC_SendSelect         CC_YES
#define CC_Quote                        CC_YES
#define CC_RSA_Decrypt                  (CC_YES && ALG_RSA)
#define CC_RSA_Encrypt                  (CC_YES && ALG_RSA)
#define CC_ReadClock                    CC_YES
#define CC_ReadPublic                   CC_YES
#define CC_Rewrap                       CC_YES
#define CC_SelfTest                     CC_YES
#define CC_SequenceComplete             CC_YES
#define CC_SequenceUpdate               CC_YES
#define CC_SetAlgorithmSet              CC_YES
#define CC_SetCommandCodeAuditStatus    CC_YES
#define CC_SetPrimaryPolicy             CC_YES
#define CC_Shutdown                     CC_YES
#define CC_Sign                         CC_YES
#define CC_StartAuthSession             CC_YES
#define CC_Startup                      CC_YES
#define CC_StirRandom                   CC_YES
#define CC_TestParms                    CC_YES
#define CC_Unseal                       CC_YES
#define CC_Vendor_TCG_Test              CC_YES
#define CC_VerifySignature              CC_YES
#define CC_ZGen_2Phase                  (CC_YES && ALG_ECC)
#define CC_NV_DefineSpace2              CC_YES
#define CC_NV_ReadPublic2               CC_YES
#define CC_SetCapability                CC_NO

#endif  // _TPM_PROFILE_COMMAND_LIST_H_
```

/TpmConfiguration/TpmConfiguration/TpmProfile_Common.h

```
// clang-format off
// clang-format off to preserve define alignment breaking sections.

// this file defines the common optional selections for the TPM library build
// Requires basic YES/NO defines are already set (by TpmBuildSwitches.h)
// Less frequently changed items are in other TpmProfile Headers.

#ifndef _TPM_PROFILE_COMMON_H_
#define _TPM_PROFILE_COMMON_H_
// YES & NO defined by TpmBuildSwitches.h
#if (YES != 1 || NO != 0)
#  error YES or NO incorrectly set
#endif
#if defined(ALG_YES) || defined(ALG_NO)
#  error ALG_YES and ALG_NO should only be defined by the TpmProfile_Common.h file
#endif

// Change these definitions to turn all algorithms ON or OFF. That is, to turn
// all algorithms on, set ALG_NO to YES. This is intended as a debug feature.
#define  ALG_YES                    YES
#define  ALG_NO                     NO

// Defines according to the processor being built for.
// Are building for a BIG_ENDIAN processor?
#define  BIG_ENDIAN_TPM             NO
#define  LITTLE_ENDIAN_TPM          !BIG_ENDIAN_TPM
// Does the processor put the most-significant bit at bit position 0?
#define MOST_SIGNIFICANT_BIT_0      NO
#define LEAST_SIGNIFICANT_BIT_0     !MOST_SIGNIFICANT_BIT_0
// Does processor support Auto align?
#define  AUTO_ALIGN                 NO

//**********************************************
```

```
// Defines for Symmetric Algorithms
//***********************************************

#define ALG_AES                     ALG_YES

#define     AES_128                     (YES * ALG_AES)
#define     AES_192                     (NO  * ALG_AES)
#define     AES_256                     (YES * ALG_AES)

#define ALG_SM4                     ALG_NO

#define     SM4_128                     (NO  * ALG_SM4)

#define ALG_CAMELLIA                ALG_YES

#define     CAMELLIA_128                (YES * ALG_CAMELLIA)
#define     CAMELLIA_192                (NO  * ALG_CAMELLIA)
#define     CAMELLIA_256                (YES * ALG_CAMELLIA)

// must be yes if any above are yes.
#define ALG_SYMCIPHER               (ALG_AES || ALG_SM4 || ALG_CAMELLIA)
#define ALG_CMAC                    (YES * ALG_SYMCIPHER)

// block cipher modes
#define ALG_CTR                     ALG_YES
#define ALG_OFB                     ALG_YES
#define ALG_CBC                     ALG_YES
#define ALG_CFB                     ALG_YES
#define ALG_ECB                     ALG_YES

//***********************************************
// Defines for RSA Asymmetric Algorithms
//***********************************************
#define ALG_RSA                     ALG_YES
#define     RSA_1024                        (YES * ALG_RSA)
#define     RSA_2048                        (YES * ALG_RSA)
#define     RSA_3072                        (YES * ALG_RSA)
#define     RSA_4096                        (YES * ALG_RSA)
#define     RSA_16384                       (NO  * ALG_RSA)

#define     ALG_RSASSA                      (YES * ALG_RSA)
#define     ALG_RSAES                       (YES * ALG_RSA)
#define     ALG_RSAPSS                      (YES * ALG_RSA)
#define     ALG_OAEP                        (YES * ALG_RSA)

// RSA Implementation Styles
// use Chinese Remainder Theorem (5 prime) format for private key ?
#define CRT_FORMAT_RSA                  YES
#define RSA_DEFAULT_PUBLIC_EXPONENT     0x00010001

//***********************************************
// Defines for ECC Asymmetric Algorithms
//***********************************************
#define ALG_ECC                     ALG_YES
#define     ALG_ECDH                        (YES * ALG_ECC)
#define     ALG_ECDSA                       (YES * ALG_ECC)
#define     ALG_ECDAA                       (YES * ALG_ECC)
#define     ALG_SM2                         (YES * ALG_ECC)
#define     ALG_ECSCHNORR                   (YES * ALG_ECC)
#define     ALG_ECMQV                       (YES * ALG_ECC)
#define     ALG_KDF1_SP800_56A              (YES * ALG_ECC)
#define     ALG_EDDSA                       (NO  * ALG_ECC)
#define     ALG_EDDSA_PH                    (NO  * ALG_ECC)

#define     ECC_NIST_P192                   (YES * ALG_ECC)
#define     ECC_NIST_P224                   (YES * ALG_ECC)
```

```
#define     ECC_NIST_P256                   (YES * ALG_ECC)
#define     ECC_NIST_P384                   (YES * ALG_ECC)
#define     ECC_NIST_P521                   (YES * ALG_ECC)
#define     ECC_BN_P256                     (YES * ALG_ECC)
#define     ECC_BN_P638                     (YES * ALG_ECC)
#define     ECC_SM2_P256                    (YES * ALG_ECC)

#define     ECC_BP_P256_R1                  (NO * ALG_ECC)
#define     ECC_BP_P384_R1                  (NO * ALG_ECC)
#define     ECC_BP_P512_R1                  (NO * ALG_ECC)
#define     ECC_CURVE_25519                 (NO * ALG_ECC)
#define     ECC_CURVE_448                   (NO * ALG_ECC)

//**********************************************
// Defines for Hash/XOF Algorithms
//**********************************************
#define ALG_MGF1                            ALG_YES
#define ALG_SHA1                            ALG_YES
#define ALG_SHA256                          ALG_YES
#define ALG_SHA256_192                      ALG_NO
#define ALG_SHA384                          ALG_YES
#define ALG_SHA512                          ALG_NO

#define ALG_SHA3_256                        ALG_NO
#define ALG_SHA3_384                        ALG_NO
#define ALG_SHA3_512                        ALG_NO

#define ALG_SM3_256                         ALG_NO

#define ALG_SHAKE256_192                    ALG_NO
#define ALG_SHAKE256_256                    ALG_NO
#define ALG_SHAKE256_512                    ALG_NO

//**********************************************
// Defines for Stateful Signature Algorithms
//**********************************************
#define ALG_LMS                             ALG_NO
#define ALG_XMSS                            ALG_NO

//**********************************************
// Defines for Keyed Hashes
//**********************************************
#define ALG_KEYEDHASH                       ALG_YES
#define ALG_HMAC                            ALG_YES

//**********************************************
// Defines for KDFs
//**********************************************
#define ALG_KDF2                            ALG_YES
#define ALG_KDF1_SP800_108                  ALG_YES

//**********************************************
// Defines for Obscuration/MISC/compatibility
//**********************************************
#define ALG_XOR                             ALG_YES

//**********************************************
// Defines controlling ACT
//**********************************************
#define ACT_SUPPORT                         YES
#define RH_ACT_0                                (YES * ACT_SUPPORT)
#define RH_ACT_1                                ( NO * ACT_SUPPORT)
#define RH_ACT_2                                ( NO * ACT_SUPPORT)
#define RH_ACT_3                                ( NO * ACT_SUPPORT)
#define RH_ACT_4                                ( NO * ACT_SUPPORT)
#define RH_ACT_5                                ( NO * ACT_SUPPORT)
```

```
#define RH_ACT_6                                ( NO * ACT_SUPPORT)
#define RH_ACT_7                                ( NO * ACT_SUPPORT)
#define RH_ACT_8                                ( NO * ACT_SUPPORT)
#define RH_ACT_9                                ( NO * ACT_SUPPORT)
#define RH_ACT_A                                (YES * ACT_SUPPORT)
#define RH_ACT_B                                ( NO * ACT_SUPPORT)
#define RH_ACT_C                                ( NO * ACT_SUPPORT)
#define RH_ACT_D                                ( NO * ACT_SUPPORT)
#define RH_ACT_E                                ( NO * ACT_SUPPORT)
#define RH_ACT_F                                ( NO * ACT_SUPPORT)


//**********************************************
// Enable VENDOR_PERMANENT_AUTH_HANDLE?
//**********************************************
#define VENDOR_PERMANENT_AUTH_ENABLED       NO
// if YES, this must be valid per Part2 (TPM_RH_AUTH_00 - TPM_RH_AUTH_FF)
// if NO, this must be #undef
#undef  VENDOR_PERMANENT_AUTH_HANDLE

//**********************************************
// Defines controlling optional implementation
//**********************************************
#define FIELD_UPGRADE_IMPLEMENTED           NO

//**********************************************
// Buffer Sizes based on implementation
//**********************************************
// When using PC CRB, the page size for both commands and
// control registers is 4k.  The command buffer starts at
// offset 0x80, so the net size available is:
#define  MAX_COMMAND_SIZE               (4096-0x80)
#define  MAX_RESPONSE_SIZE              (4096-0x80)

//**********************************************
// Vendor Info
//**********************************************
// max buffer for vendor commands
// Max data buffer leaving space for TPM2B size prefix
#define VENDOR_COMMAND_COUNT          0
#define MAX_VENDOR_BUFFER_SIZE          (MAX_RESPONSE_SIZE-2)
#define PRIVATE_VENDOR_SPECIFIC_BYTES RSA_PRIVATE_SIZE

//**********************************************
// Defines controlling Firmware- and SVN-limited objects
//**********************************************
#define FW_LIMITED_SUPPORT                    YES
#define SVN_LIMITED_SUPPORT                   YES

//**********************************************
// Defines controlling External NV
//**********************************************
// This is a software reference implementation of the TPM: there is no
// "external NV" as such. This #define configures the TPM to implement
// "external NV" that is stored in the same place as "internal NV."
// NOTE: enabling this doesn't necessarily mean that the expanded
// (external-NV-specific) attributes are supported.
#define EXTERNAL_NV                           YES

#endif // _TPM_PROFILE_COMMON_H_
```

/TpmConfiguration/TpmConfiguration/TpmProfile_ErrorCodes.h

```
//** Introduction
// This file defines error codes used in failure macros in the TPM Core Library.
```

```
// This file is part of TpmConfiguration because the Platform library can add error
// codes of it's own, and ultimately the specific error codes are a vendor decision
// because TPM2_GetTestResult returns manufacturer-defined data in failure mode.
// The only thing in this file that must be consistent with a vendor's implementation
// are the _names_ of error codes used by the core library.  Even the values can
// change and are only a suggestion.

#ifndef _TPMPROFILE_ERRORCODES_H
#define _TPMPROFILE_ERRORCODES_H

// turn off clang-format because alignment doesn't persist across comments
// with current settings
// clang-format off

#define FATAL_ERROR_ALLOCATION       (1)
#define FATAL_ERROR_DIVIDE_ZERO      (2)
#define FATAL_ERROR_INTERNAL         (3)
#define FATAL_ERROR_PARAMETER        (4)
#define FATAL_ERROR_ENTROPY          (5)
#define FATAL_ERROR_SELF_TEST        (6)
#define FATAL_ERROR_CRYPTO           (7)
#define FATAL_ERROR_NV_UNRECOVERABLE (8)

// indicates that the TPM has been re-manufactured after an
// unrecoverable NV error
#define FATAL_ERROR_REMANUFACTURED   (9)
#define FATAL_ERROR_DRBG             (10)
#define FATAL_ERROR_MOVE_SIZE        (11)
#define FATAL_ERROR_COUNTER_OVERFLOW (12)
#define FATAL_ERROR_SUBTRACT         (13)
#define FATAL_ERROR_MATHLIBRARY      (14)
// end of codes defined through v1.52

// leave space for numbers that may have been used by vendors or platforms.
// Ultimately this file and these ranges are only a suggestion because
// TPM2_GetTestResult returns manufacturer-defined data in failure mode.
// Reserve 15-499
#define FATAL_ERROR_RESERVED_START   (15)
#define FATAL_ERROR_RESERVED_END     (499)

// Additional error codes defined by TPM library:
#define FATAL_ERROR_ASSERT           (500)
// Platform library violated interface contract.
#define FATAL_ERROR_PLATFORM         (600)

// Test/Simulator errors 1000+
#define FATAL_ERROR_FORCED           (1000)

#endif  // _TPMPROFILE_ERRORCODES_H
```

/TpmConfiguration/TpmConfiguration/TpmProfile_Misc.h

```
// Misc profile settings that don't currently have a better home.
// These are rarely changed, but available for vendor customization.

#ifndef _TPM_PROFILE_MISC_H_
#define _TPM_PROFILE_MISC_H_

// YES & NO defined by TpmBuildSwitches.h
#if(YES != 1 || NO != 0)
#  error YES or NO incorrectly set
#endif

// clang-format off
// clang-format off to preserve horizontal spacing
```

```
#define IMPLEMENTATION_PCR          24
#define PLATFORM_PCR                24
#define DRTM_PCR                    17
#define HCRTM_PCR                   0
#define NUM_LOCALITIES              5
#define MAX_HANDLE_NUM              3
#define MAX_ACTIVE_SESSIONS         64
#define MAX_LOADED_SESSIONS         3
#define MAX_SESSION_NUM             3
#define MAX_LOADED_OBJECTS          3
#define MIN_EVICT_OBJECTS           2
#define NUM_POLICY_PCR_GROUP        1
#define NUM_AUTHVALUE_PCR_GROUP     1
#define MAX_CONTEXT_SIZE            2168
#define MAX_DIGEST_BUFFER           1024
#define MAX_NV_INDEX_SIZE           2048
#define MAX_NV_BUFFER_SIZE          1024
#define MAX_CAP_BUFFER              1024
#define NV_MEMORY_SIZE              16384
#define MIN_COUNTER_INDICES         8
#define NUM_STATIC_PCR              16
#define MAX_ALG_LIST_SIZE           64
#define PRIMARY_SEED_SIZE           32
#define CONTEXT_ENCRYPT_ALGORITHM   AES
#define NV_CLOCK_UPDATE_INTERVAL    22
#define NUM_POLICY_PCR              1

#define ORDERLY_BITS                8
#define MAX_SYM_DATA                128
#define MAX_RNG_ENTROPY_SIZE        64
#define RAM_INDEX_SPACE             512
#define ENABLE_PCR_NO_INCREMENT     YES

#define SIZE_OF_X509_SERIAL_NUMBER 20

// amount of space the platform can provide in PERSISTENT_DATA during
// manufacture
#define PERSISTENT_DATA_PLATFORM_SPACE  16

// structure padding space for these structures.  Used if a
// particular configuration needs them to be aligned to a
// specific size
#define ORDERLY_DATA_PADDING            0
#define STATE_CLEAR_DATA_PADDING        0
#define STATE_RESET_DATA_PADDING        0

// configuration values that may vary by SIMULATION/DEBUG
#if SIMULATION && DEBUG
// This forces the use of a smaller context slot size. This reduction reduces the
// range of the epoch allowing the tester to force the epoch to occur faster than
// the normal production size
#  define CONTEXT_SLOT UINT8
#else
#  define CONTEXT_SLOT UINT16
#endif

#endif  // _TPM_PROFILE_MISC_H_
```

/TpmConfiguration/TpmConfiguration/VendorInfo.h

```
#ifndef _VENDORINFO_H
#define _VENDORINFO_H

// Define the TPM specification-specific capability values.
#define TPM_SPEC_FAMILY      (0x322E3000)
```

```
#define TPM_SPEC_LEVEL       (00)
#define TPM_SPEC_VERSION     (183)
#define TPM_SPEC_YEAR        (2024)
#define TPM_SPEC_DAY_OF_YEAR (25)
#define MAX_VENDOR_PROPERTY  (1)

// Define the platform specification-specific capability values.
#define PLATFORM_FAMILY      (0)
#define PLATFORM_LEVEL       (0)
#define PLATFORM_VERSION     (0)
#define PLATFORM_YEAR        (0)
#define PLATFORM_DAY_OF_YEAR (0)

#endif
```

Annex B
(informative)
Library-Specific

B.1 Introduction

This clause contains the files that are specific to a cryptographic library used by the TPM code.

Three categories are defined for cryptographic functions:

1) big number math (asymmetric cryptography),

2) symmetric ciphers, and

3) hash functions.

The code is structured to make it possible to use different libraries for different categories. For example, one might choose to use OpenSSL for its math library, but use a different library for hashing and symmetric cryptography. Since OpenSSL supports all three categories, it might be more typical to combine libraries of specific functions; that is, one library might only contain block ciphers while another supports big number math.

B.2 Common Cryptographic Functionality

/tpm/cryptolibs/common/include/CryptoInterface.h

```
//** Introduction
//
// This file contains prototypes that are common to all TPM crypto interfaces.
//
#ifndef CRYPTO_INTERFACE_H
#define CRYPTO_INTERFACE_H

#include "TpmConfiguration/TpmBuildSwitches.h"

#if SIMULATION && CRYPTO_LIB_REPORTING

typedef struct crypto_impl_description
{
    // The name of the crypto library, ASCII encoded.
    char name[32];
    // The version of the crypto library, ASCII encoded.
    char version[32];
} _CRYPTO_IMPL_DESCRIPTION;

// When building the simulator, the plugged-in crypto libraries can report its
// version information by implementing these interfaces.
void _crypto_GetSymImpl(_CRYPTO_IMPL_DESCRIPTION* result);
void _crypto_GetHashImpl(_CRYPTO_IMPL_DESCRIPTION* result);
void _crypto_GetMathImpl(_CRYPTO_IMPL_DESCRIPTION* result);

#endif  // SIMULATION && CRYPTO_LIB_REPORTING

#endif  // CRYPTO_INTERFACE_H
```

/tpm/cryptolibs/common/include/MathLibraryInterface.h

```
//** Introduction
//
// This file contains the function prototypes for the functions that need to be
// present in the selected math library. For each function listed, there should
```

```
// be a small stub function. That stub provides the interface between the TPM
// code and the support library. In most cases, the stub function will only need
// to do a format conversion between the Crypt_* formats to the internal support
// library format.  Since the external library also provides the buffer macros
// for the underlying types, this is typically just a cast from the TPM type to
// the internal type.
//
// Arithmetic operations return a BOOL to indicate if the operation completed
// successfully or not.

#ifndef MATH_LIBRARY_INTERFACE_H
#define MATH_LIBRARY_INTERFACE_H

// Types
#include "MathLibraryInterfaceTypes.h"

// ***************************************************************************
// Library Level Functions
// ***************************************************************************

//** ExtMath_LibInit()
// This function is called by CryptInit() so that necessary initializations can be
// performed on the cryptographic library.
LIB_EXPORT int ExtMath_LibInit(void);

//** MathLibraryCompatibililtyCheck()
// This function is only used during development to make sure that the library
// that is being referenced is using the same size of data structures as the TPM.
LIB_EXPORT BOOL ExtMath_Debug_CompatibilityCheck(void);

// ***************************************************************************
// Integer/Number Functions (non-ECC)
// ***************************************************************************

// ##################
// type initializers
// ##################

//** ExtMath_Initialize_Int()
// Initialize* functions tells the Crypt_Int types how large of a value it can
// contain which is a compile time constant
LIB_EXPORT Crypt_Int* ExtMath_Initialize_Int(Crypt_Int* buffer, NUMBYTES bits);

// ##################
// Buffer Converters
// ##################
// convert TPM2B byte datainto the private format. The Crypt_Int must already be
// initialized with it's maximum size. Byte-based Initializers must be MSB first
// (TPM external format).
LIB_EXPORT Crypt_Int* ExtMath_IntFromBytes(
    Crypt_Int* buffer, const BYTE* input, NUMBYTES byteCount);
// Convert Crypt_Int into external format as a byte array.
LIB_EXPORT BOOL ExtMath_IntToBytes(
    const Crypt_Int* value, BYTE* output, NUMBYTES* pByteCount);
// Set Crypt_Int to a given small value. Words are native format.
LIB_EXPORT Crypt_Int* ExtMath_SetWord(Crypt_Int* buffer, crypt_uword_t word);

// ##################
// Copy Functions
// ##################

//*** ExtMath_Copy()
// Function to copy a bignum_t. If the output is NULL, then
// nothing happens. If the input is NULL, the output is set to zero.
LIB_EXPORT BOOL ExtMath_Copy(Crypt_Int* out, const Crypt_Int* in);
```

```
// ################################
// Ordinary Arithmetic, writ large
// ################################

//** ExtMath_Multiply()
// Multiplies two numbers and returns the result
LIB_EXPORT BOOL ExtMath_Multiply(
    Crypt_Int* result, const Crypt_Int* multiplicand, const Crypt_Int* multiplier);

//** ExtMath_Divide()
// This function divides two Crypt_Int* values. The function returns FALSE if there is
// an error in the operation. Quotient may be null, in which case this function
returns
// only the remainder.
LIB_EXPORT BOOL ExtMath_Divide(Crypt_Int*       quotient,
                               Crypt_Int*       remainder,
                               const Crypt_Int* dividend,
                               const Crypt_Int* divisor);

//** ExtMath_GCD()
// Get the greatest common divisor of two numbers. This function is only needed
// when the TPM implements RSA.
LIB_EXPORT BOOL ExtMath_GCD(
    Crypt_Int* gcd, const Crypt_Int* number1, const Crypt_Int* number2);

//*** ExtMath_Add()
// This function adds two Crypt_Int* values. This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_Add(
    Crypt_Int* result, const Crypt_Int* op1, const Crypt_Int* op2);

//*** ExtMath_AddWord()
// This function adds a word value to a Crypt_Int*. This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_AddWord(
    Crypt_Int* result, const Crypt_Int* op, crypt_uword_t word);

//*** ExtMath_Subtract()
// This function does subtraction of two Crypt_Int* values and returns result = op1 -
op2
// when op1 is greater than op2. If op2 is greater than op1, then a fault is
// generated. This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_Subtract(
    Crypt_Int* result, const Crypt_Int* op1, const Crypt_Int* op2);

//*** ExtMath_SubtractWord()
// This function subtracts a word value from a Crypt_Int*. This function always
// returns TRUE.
LIB_EXPORT BOOL ExtMath_SubtractWord(
    Crypt_Int* result, const Crypt_Int* op, crypt_uword_t word);

// ################################
// Modular Arithmetic, writ large
// ################################

//** ExtMath_Mod()
// compute valueAndResult = valueAndResult mod modulus
// This function divides two Crypt_Int* values and returns only the remainder,
// replacing the original dividend. The function returns FALSE if there is an
// error in the operation.
LIB_EXPORT BOOL ExtMath_Mod(Crypt_Int* valueAndResult, const Crypt_Int* modulus);

//** ExtMath_ModMult()
// Compute result = (op1 * op2) mod modulus
LIB_EXPORT BOOL ExtMath_ModMult(Crypt_Int*       result,
                                const Crypt_Int* op1,
                                const Crypt_Int* op2,
                                const Crypt_Int* modulus);
```

```
//** ExtMath_ModExp()
// Compute result = (number ^ exponent) mod modulus
// where ^ indicates exponentiation.
// This function is only needed when the TPM implements RSA.
LIB_EXPORT BOOL ExtMath_ModExp(Crypt_Int*       result,
                               const Crypt_Int* number,
                               const Crypt_Int* exponent,
                               const Crypt_Int* modulus);

//** ExtMath_ModInverse()
// Compute the modular multiplicative inverse.
// result = (number ^ -1) mod modulus
// This function is only needed when the TPM implements RSA.
LIB_EXPORT BOOL ExtMath_ModInverse(
    Crypt_Int* result, const Crypt_Int* number, const Crypt_Int* modulus);

//** ExtMath_ModInversePrime()
// Compute the modular multiplicative inverse. This is an optimized function for
// the case where the modulus is known to be prime.
//
// CAUTION: Depending on the library implementation this may be much faster than
// the normal ModInverse, and therefore is subject to exposing the fact the
// modulus is prime via a timing side-channel. In many cases (e.g. ECC primes),
// the prime is not sensitive and this optimized route can be used.
LIB_EXPORT BOOL ExtMath_ModInversePrime(
    Crypt_Int* result, const Crypt_Int* number, const Crypt_Int* primeModulus);

//*** ExtMath_ModWord()
// compute numerator
// This function does modular division of a big number when the modulus is a
// word value.
LIB_EXPORT crypt_word_t ExtMath_ModWord(const Crypt_Int* numerator,
                                        crypt_word_t     modulus);

// ################################
// Queries
// ################################

//*** ExtMath_UnsignedCmp()
// This function performs a comparison of op1 to op2. The compare is approximately
// constant time if the size of the values used in the compare is consistent
// across calls (from the same line in the calling code).
//  Return Type: int
//      < 0             op1 is less than op2
//      0               op1 is equal to op2
//      > 0             op1 is greater than op2
LIB_EXPORT int ExtMath_UnsignedCmp(const Crypt_Int* op1, const Crypt_Int* op2);

//*** ExtMath_UnsignedCmpWord()
// Compare a Crypt_Int* to a crypt_uword_t.
//  Return Type: int
//      -1              op1 is less that word
//      0               op1 is equal to word
//      1               op1 is greater than word
LIB_EXPORT int ExtMath_UnsignedCmpWord(const Crypt_Int* op1, crypt_uword_t word);

//*** ExtMath_IsEqualWord()
// Compare a Crypt_Int* to a crypt_uword_t for equality
//  Return Type: BOOL
LIB_EXPORT BOOL ExtMath_IsEqualWord(const Crypt_Int* bn, crypt_uword_t word);

//*** ExtMath_IsZero()
// Compare a Crypt_Int* to zero, expected to be O(1) time.
//  Return Type: BOOL
LIB_EXPORT BOOL ExtMath_IsZero(const Crypt_Int* op1);
```

```
//*** ExtMath_MostSigBitNum()
//
// This function returns the zero-based number of the MSb (Most significant bit)
// of a Crypt_Int* value.
//
// Return Type: int
//
//      -1              the word was zero or 'bn' was NULL
//      n               the bit number of the most significant bit in the word
LIB_EXPORT int ExtMath_MostSigBitNum(const Crypt_Int* bn);

//*** ExtMath_GetLeastSignificant32bits()
//
// This function returns the least significant 32-bits of an integer value
// Return Type: uint32_t
LIB_EXPORT uint32_t ExtMath_GetLeastSignificant32bits(const Crypt_Int* bn);

//*** ExtMath_SizeInBits()
//
// This function returns the number of bits required to hold a number. It is one
// greater than the Msb.  This function is expected to be side channel safe, and
// may be O(size) or O(1) where 'size' is the allocated (not actual) size of the
// value.
LIB_EXPORT unsigned ExtMath_SizeInBits(const Crypt_Int* n);

// ################################
// Bitwise Operations
// ################################

//*** ExtMath_SetBit()
//
// This function will SET a bit in a Crypt_Int*. Bit 0 is the least-significant
// bit in the 0th digit_t. The function returns TRUE if the bitNum is within the
// range valid for the given number.  If bitNum is too large, the function
// should return FALSE, and the TPM will enter failure mode.
// Return Type: BOOL
LIB_EXPORT BOOL ExtMath_SetBit(Crypt_Int*   bn,     // IN/OUT: big number to modify
                               unsigned int bitNum  // IN: Bit number to SET
);

//*** ExtMath_TestBit()
// This function is used to check to see if a bit is SET in a bignum_t. The 0th bit
// is the LSb of d[0].
//  Return Type: BOOL
//      TRUE(1)         the bit is set
//      FALSE(0)        the bit is not set or the number is out of range
LIB_EXPORT BOOL ExtMath_TestBit(Crypt_Int*   bn,     // IN: number to check
                                unsigned int bitNum  // IN: bit to test
);

//***ExtMath_MaskBits()
// This function is used to mask off high order bits of a big number.
// The returned value will have no more than 'maskBit' bits
// set.
// Note: There is a requirement that unused words of a bignum_t are set to zero.
//  Return Type: BOOL
//      TRUE(1)         result masked
//      FALSE(0)        the input was not as large as the mask
LIB_EXPORT BOOL ExtMath_MaskBits(
    Crypt_Int*    bn,      // IN/OUT: number to mask
    crypt_uword_t maskBit  // IN: the bit number for the mask.
);

//*** ExtMath_ShiftRight()
// This function will shift a Crypt_Int* to the right by the shiftAmount.
```

```
// This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_ShiftRight(
    Crypt_Int* result, const Crypt_Int* toShift, uint32_t shiftAmount);

// ***************************************************************************
// ECC Functions
// ***************************************************************************
// ##################
// type initializers
// ##################

//** initialize point structure given memory size of each coordinate
LIB_EXPORT Crypt_Point* ExtEcc_Initialize_Point(Crypt_Point* buffer,
                                                NUMBYTES     bitsPerCoord);

//** ExtEcc_CurveInitialize()
// This function is used to initialize a Crypt_EccCurve structure. The
// structure is a set of pointers to Crypt_Int* values. The curve-dependent values are
// set by a different function. This function is only needed
// if the TPM supports ECC.
LIB_EXPORT const Crypt_EccCurve* ExtEcc_CurveInitialize(Crypt_EccCurve* E,
                                                        TPM_ECC_CURVE   curveId);

// ##################
// DESTRUCTOR - See Warning
// ##################

//*** ExtEcc_CurveFree()
// This function will free the allocated components of the curve and end the
// frame in which the curve data exists.
// WARNING: Not guaranteed to be called in presence of LONGJMP.
LIB_EXPORT void ExtEcc_CurveFree(const Crypt_EccCurve* E);

// ##################
// Buffer Converters
// ##################
//** point structure to/from raw coordinate buffers.
LIB_EXPORT Crypt_Point* ExtEcc_PointFromBytes(Crypt_Point* buffer,
                                              const BYTE*  x,
                                              NUMBYTES     nBytesX,
                                              const BYTE*  y,
                                              NUMBYTES     nBytesY);

LIB_EXPORT BOOL         ExtEcc_PointToBytes(
            const Crypt_Point* point, BYTE* x, NUMBYTES* nBytesX, BYTE* y, NUMBYTES*
nBytesY);

// ####################
// ECC Point Operations
// ####################

//** ExtEcc_PointMultiply()
// This function does a point multiply of the form R = [d]S. A return of FALSE
// indicates that the result was the point at infinity. This function is only needed
// if the TPM supports ECC.
LIB_EXPORT BOOL ExtEcc_PointMultiply(Crypt_Point*          R,
                                     const Crypt_Point*    S,
                                     const Crypt_Int*      d,
                                     const Crypt_EccCurve* E);

//** ExtEcc_PointMultiplyAndAdd()
// This function does a point multiply of the form R = [d]S + [u]Q. A return of
// FALSE indicates that the result was the point at infinity. This function is only
// needed if the TPM supports ECC.
LIB_EXPORT BOOL ExtEcc_PointMultiplyAndAdd(Crypt_Point*          R,
                                           const Crypt_Point*    S,
```

```
                                        const Crypt_Int*      d,
                                        const Crypt_Point*    Q,
                                        const Crypt_Int*      u,
                                        const Crypt_EccCurve* E);

//** ExtEcc_PointAdd()
// This function does a point add R = S + Q. A return of FALSE
// indicates that the result was the point at infinity. This function is only needed
// if the TPM supports ECC.
LIB_EXPORT BOOL ExtEcc_PointAdd(Crypt_Point*          R,
                                const Crypt_Point*    S,
                                const Crypt_Point*    Q,
                                const Crypt_EccCurve* E);

// ######################
// ECC Point Information
// ######################
LIB_EXPORT BOOL ExtEcc_IsPointOnCurve(const Crypt_Point* Q, const Crypt_EccCurve* E);
LIB_EXPORT BOOL ExtEcc_IsInfinityPoint(const Crypt_Point* pt);
// extract the X-Coordinate of a point
LIB_EXPORT const Crypt_Int* ExtEcc_PointX(const Crypt_Point* pt);

// extract the Y-Coordinate of a point
// (no current use case for the Y coordinate alone, signatures use X)
// LIB_EXPORT const Crypt_Int* ExtEcc_PointY(const Crypt_Point* pt);

// ######################
// ECC Curve Information
// ######################
// These functions are expected to be fast, returning pre-built constants without
// allocation or copying.
LIB_EXPORT const Crypt_Int*   ExtEcc_CurveGetPrime(TPM_ECC_CURVE curveId);
LIB_EXPORT const Crypt_Int*   ExtEcc_CurveGetOrder(TPM_ECC_CURVE curveId);
LIB_EXPORT const Crypt_Int*   ExtEcc_CurveGetCofactor(TPM_ECC_CURVE curveId);
LIB_EXPORT const Crypt_Int*   ExtEcc_CurveGet_a(TPM_ECC_CURVE curveId);
LIB_EXPORT const Crypt_Int*   ExtEcc_CurveGet_b(TPM_ECC_CURVE curveId);
LIB_EXPORT const Crypt_Point* ExtEcc_CurveGetG(TPM_ECC_CURVE curveId);
LIB_EXPORT const Crypt_Int*   ExtEcc_CurveGetGx(TPM_ECC_CURVE curveId);
LIB_EXPORT const Crypt_Int*   ExtEcc_CurveGetGy(TPM_ECC_CURVE curveId);
LIB_EXPORT TPM_ECC_CURVE      ExtEcc_CurveGetCurveId(const Crypt_EccCurve* E);

#endif
```

/tpm/cryptolibs/common/include/MathLibraryInterfaceTypes.h

```
//** Introduction
// This file contains the declaration and initialization macros for
// low-level cryptographic buffer types.  This requires the underlying
// Crypto library to have already defined the CRYPT_INT_BUF family of
// macros.  See tpm_crypto_lib.md for details.

#ifndef MATH_LIBRARY_INTERFACE_TYPES_H
#define MATH_LIBRARY_INTERFACE_TYPES_H

#ifndef CRYPT_INT_BUF
#  error CRYPT_INT_BUF must be defined before including this file.
#endif
#ifndef CRYPT_POINT_BUF
#  error CRYPT_POINT_BUF must be defined before including this file.
#endif
#ifndef CRYPT_CURVE_BUF
#  error CRYPT_CURVE_BUF must be defined before including this file.
#endif

// Crypt_Int underlying types Crypt_Int is an abstract type that is used as a
```

```
// pointer. The underlying math library is expected to be able to find the
// actual allocated size for a given Crypt_Int object given a pointer to it, and
// therefore we typedef here to a size 1 (smallest possible).
typedef CRYPT_INT_BUF(one, 1) Crypt_Int;
typedef CRYPT_POINT_BUF(pointone, 1) Crypt_Point;
typedef CRYPT_CURVE_BUF(curvebuft, MAX_ECC_KEY_BITS) Crypt_EccCurve;

// produces bare typedef ci_<typename>_t
#define CRYPT_INT_TYPE(typename, bits) \
    typedef CRYPT_INT_BUF(ci_##typename##_buf_t, bits) ci_##typename##_t

// produces allocated `Crypt_Int* varname` backed by a
// stack buffer named `<varname>_buf`.  Initialization at the discretion of the
// ExtMath library.
#define CRYPT_INT_VAR(varname, bits)                          \
    CRYPT_INT_BUF(ci_##varname##_buf_t, bits) varname##_buf; \
    Crypt_Int* varname = ExtMath_Initialize_Int((Crypt_Int*)&(varname##_buf), bits);

// produces initialized `Crypt_Int* varname = (TPM2B) initializer` backed by a
// stack buffer named `<varname>_buf`
#define CRYPT_INT_INITIALIZED(varname, bits, initializer)          \
    CRYPT_INT_BUF(cibuf##varname, bits) varname##_buf;             \
    Crypt_Int* varname = TpmMath_IntFrom2B(                        \
        ExtMath_Initialize_Int((Crypt_Int*)&(varname##_buf), bits), \
        (TPM2B*)initializer);

// convenience variants of above:
// largest supported integer
#define CRYPT_INT_MAX(varname) CRYPT_INT_VAR(varname, LARGEST_NUMBER_BITS)

#define CRYPT_INT_MAX_INITIALIZED(name, initializer) \
    CRYPT_INT_INITIALIZED(name, LARGEST_NUMBER_BITS, initializer)

// A single RADIX_BITS value.
#define CRYPT_INT_WORD(name) CRYPT_INT_VAR(name, RADIX_BITS)

#define CRYPT_INT_WORD_INITIALIZED(varname, initializer)                \
    CRYPT_INT_BUF(cibuf##varname, RADIX_BITS) varname##_buf;            \
    Crypt_Int* varname = ExtMath_SetWord(                               \
        ExtMath_Initialize_Int((Crypt_Int*)&(varname##_buf), RADIX_BITS), \
        initializer);

// Crypt_EccCurve underlying types
#define CRYPT_CURVE_INITIALIZED(varname, initializer)             \
    CRYPT_CURVE_BUF(cv##varname, MAX_ECC_KEY_BITS) varname##_buf; \
    const Crypt_EccCurve* varname =                               \
        ExtEcc_CurveInitialize(&(varname##_buf), initializer)

/* no guarantee free will be called in the presence of longjmp */
#define CRYPT_CURVE_FREE(varname) ExtEcc_CurveFree(varname)

// Crypt_Point underlying types
#define CRYPT_POINT_VAR(varname)                                        \
    CRYPT_POINT_BUF(cp_##varname##_buf_t, MAX_ECC_KEY_BITS) varname##_buf; \
    Crypt_Point* varname =                                              \
        ExtEcc_Initialize_Point((Crypt_Point*)&(varname##_buf), MAX_ECC_KEY_BITS);

#define CRYPT_POINT_INITIALIZED(varname, initValue)                          \
    CRYPT_POINT_BUF(cp_##varname##_buf_t, MAX_ECC_KEY_BITS) varname##_buf;   \
    Crypt_Point* varname = TpmEcc_PointFrom2B(                               \
        ExtEcc_Initialize_Point((Crypt_Point*)&(varname##_buf), MAX_ECC_KEY_BITS), \
        initValue);

#endif  //MATH_LIBRARY_INTERFACE_TYPES_H
```

B.3 TpmBigNum

/tpm/cryptolibs/TpmBigNum/BnConvert.c

```
//** Introduction
// This file contains the basic conversion functions that will convert TPM2B
// to/from the internal format. The internal format is a bigNum,
//

//** Includes

#include "TpmBigNum.h"

//** Functions

//*** BnFromBytes()
// This function will convert a big-endian byte array to the internal number
// format. If bn is NULL, then the output is NULL. If bytes is null or the
// required size is 0, then the output is set to zero
LIB_EXPORT bigNum BnFromBytes(bigNum bn, const BYTE* bytes, NUMBYTES nBytes)
{
    const BYTE*   pFrom;  // 'p' points to the least significant bytes of source
    BYTE*         pTo;    // points to least significant bytes of destination
    crypt_uword_t size;
    //

    size = (bytes != NULL) ? BYTES_TO_CRYPT_WORDS(nBytes) : 0;

    // If nothing in, nothing out
    if(bn == NULL)
        return NULL;

    // make sure things fit
    pAssert(BnGetAllocated(bn) >= size);

    if(size > 0)
    {
        // Clear the topmost word in case it is not filled with data
        bn->d[size - 1] = 0;
        // Moving the input bytes from the end of the list (LSB) end
        pFrom = bytes + nBytes - 1;
        // To the LS0 of the LSW of the bigNum.
        pTo = (BYTE*)bn->d;
        for(; nBytes != 0; nBytes--)
            *pTo++ = *pFrom--;
            // For a little-endian machine, the conversion is a straight byte
            // reversal. For a big-endian machine, we have to put the words in
            // big-endian byte order
#if BIG_ENDIAN_TPM
        {
            crypt_word_t t;
            for(t = (crypt_word_t)size - 1; t >= 0; t--)
                bn->d[t] = SWAP_CRYPT_WORD(bn->d[t]);
        }
#endif
    }
    BnSetTop(bn, size);
    return bn;
}

//*** BnFrom2B()
// Convert an TPM2B to a BIG_NUM.
// If the input value does not exist, or the output does not exist, or the input
// will not fit into the output the function returns NULL
LIB_EXPORT bigNum BnFrom2B(bigNum       bn,  // OUT:
```

```
                          const TPM2B* a2B  // IN: number to convert
)
{
    if(a2B != NULL)
        return BnFromBytes(bn, a2B->buffer, a2B->size);
    // Make sure that the number has an initialized value rather than whatever
    // was there before
    BnSetTop(bn, 0);  // Function accepts NULL
    return NULL;
}

//*** BnToBytes()
// This function converts a BIG_NUM to a byte array. It converts the bigNum to a
// big-endian byte string and sets 'size' to the normalized value. If  'size' is an
// input 0, then the receiving buffer is guaranteed to be large enough for the result
// and the size will be set to the size required for bigNum (leading zeros
// suppressed).
//
// The conversion for a little-endian machine simply requires that all significant
// bytes of the bigNum be reversed. For a big-endian machine, rather than
// unpack each word individually, the bigNum is converted to little-endian words,
// copied, and then converted back to big-endian.
LIB_EXPORT BOOL BnToBytes(bigConst  bn,
                          BYTE*     buffer,
                          NUMBYTES* size  // This the number of bytes that are
                                          // available in the buffer. The result
                                          // should be this big.
)
{
    crypt_uword_t requiredSize;
    BYTE*         pFrom;
    BYTE*         pTo;
    crypt_uword_t count;
    //
    // validate inputs
    pAssert(bn && buffer && size);

    requiredSize = (BnSizeInBits(bn) + 7) / 8;
    if(requiredSize == 0)
    {
        // If the input value is 0, return a byte of zero
        *size   = 1;
        *buffer = 0;
    }
    else
    {
#if BIG_ENDIAN_TPM
        // Copy the constant input value into a modifiable value
        BN_VAR(bnL, LARGEST_NUMBER_BITS * 2);
        BnCopy(bnL, bn);
        // byte swap the words in the local value to make them little-endian
        for(count = 0; count < bnL->size; count++)
            bnL->d[count] = SWAP_CRYPT_WORD(bnL->d[count]);
        bn = (bigConst)bnL;
#endif
        if(*size == 0)
            *size = (NUMBYTES)requiredSize;
        pAssert(requiredSize <= *size);
        // Byte swap the number (not words but the whole value)
        count = *size;
        // Start from the least significant word and offset to the most significant
        // byte which is in some high word
        pFrom = (BYTE*)(&bn->d[0]) + requiredSize - 1;
        pTo   = buffer;

        // If the number of output bytes is larger than the number bytes required
```

```
        // for the input number, pad with zeros
        for(count = *size; count > requiredSize; count--)
            *pTo++ = 0;
        // Move the most significant byte at the end of the BigNum to the next most
        // significant byte position of the 2B and repeat for all significant bytes.
        for(; requiredSize > 0; requiredSize--)
            *pTo++ = *pFrom--;
    }
    return TRUE;
}

//*** BnTo2B()
// Function to convert a BIG_NUM to TPM2B.
// The TPM2B size is set to the requested 'size' which may require padding.
// If 'size' is non-zero and less than required by the value in 'bn' then an error
// is returned. If 'size' is zero, then the TPM2B is assumed to be large enough
// for the data and a2b->size will be adjusted accordingly.
LIB_EXPORT BOOL BnTo2B(bigConst bn,   // IN:
                       TPM2B*   a2B,  // OUT:
                       NUMBYTES size  // IN: the desired size
)
{
    // Set the output size
    if(bn && a2B)
    {
        a2B->size = size;
        return BnToBytes(bn, a2B->buffer, &a2B->size);
    }
    return FALSE;
}

#if ALG_ECC

//*** BnPointFromBytes()
// Function to create a BIG_POINT structure from a byte buffer in big-endian order.
// A point is going to be two ECC values in the same buffer. The values are going
// to be the size of the modulus.  They are in modular form.
LIB_EXPORT bn_point_t* BnPointFromBytes(
    bigPoint    ecP,  // OUT: the preallocated point structure
    const BYTE* x,
    NUMBYTES    nBytesX,
    const BYTE* y,
    NUMBYTES    nBytesY)
{
    if(x == NULL || y == NULL)
        return NULL;

    if(NULL != ecP)
    {
        BnFromBytes(ecP->x, x, nBytesX);
        BnFromBytes(ecP->y, y, nBytesY);
        BnSetWord(ecP->z, 1);
    }
    return ecP;
}

//*** BnPointToBytes()
// This function extracts coordinates from a BIG_POINT into
// most-significant-byte-first memory buffers (the native format of
// a TPMS_ECC_POINT.)
// on input the NUMBYTES* parameters indicate the maximum buffer size.
// on output, they represent the amount of significant data in that buffer.
LIB_EXPORT BOOL BnPointToBytes(
    pointConst ecP,  // OUT: the preallocated point structure
    BYTE*      x,
    NUMBYTES*  pBytesX,
```

```
    BYTE*       y,
    NUMBYTES*   pBytesY)
{
    pAssert(ecP && x && y && pBytesX && pBytesY);
    pAssert(BnEqualWord(ecP->z, 1));
    BOOL result = BnToBytes(ecP->x, x, pBytesX);
    result      = result && BnToBytes(ecP->y, y, pBytesY);
    // TODO: zeroize on error?
    return result;
}

#endif  // ALG_ECC
```

/tpm/cryptolibs/TpmBigNum/BnEccConstants.c

```
/*(Auto-generated)
 *  Created by TpmStructures; Version 4.4 Mar 26, 2019
 *  Date: Aug 30, 2019  Time: 02:11:52PM
 */
#include "TpmBigNum.h"
//#include "Tpm.h"
// TODO_RENAME_INC_FOLDER:private refers to the TPM_CoreLib private headers
#include <private/OIDs.h>

#if ALG_ECC

// define macros expected by EccConstantData to convert the data to BigNum format

#  define TO_ECC_64                       TO_CRYPT_WORD_64
#  define TO_ECC_56(a, b, c, d, e, f, g) TO_ECC_64(0, a, b, c, d, e, f, g)
#  define TO_ECC_48(a, b, c, d, e, f)    TO_ECC_64(0, 0, a, b, c, d, e, f)
#  define TO_ECC_40(a, b, c, d, e)       TO_ECC_64(0, 0, 0, a, b, c, d, e)
#  if RADIX_BITS > 32
#    define TO_ECC_32(a, b, c, d) TO_ECC_64(0, 0, 0, 0, a, b, c, d)
#    define TO_ECC_24(a, b, c)    TO_ECC_64(0, 0, 0, 0, 0, a, b, c)
#    define TO_ECC_16(a, b)       TO_ECC_64(0, 0, 0, 0, 0, 0, a, b)
#    define TO_ECC_8(a)           TO_ECC_64(0, 0, 0, 0, 0, 0, 0, a)
#  else  // RADIX_BITS == 32
#    define TO_ECC_32          BIG_ENDIAN_BYTES_TO_UINT32
#    define TO_ECC_24(a, b, c) TO_ECC_32(0, a, b, c)
#    define TO_ECC_16(a, b)    TO_ECC_32(0, 0, a, b)
#    define TO_ECC_8(a)        TO_ECC_32(0, 0, 0, a)
#  endif
#  define TO_ECC_192(a, b, c)                      c, b, a
#  define TO_ECC_224(a, b, c, d)                   d, c, b, a
#  define TO_ECC_256(a, b, c, d)                   d, c, b, a
#  define TO_ECC_384(a, b, c, d, e, f)             f, e, d, c, b, a
#  define TO_ECC_528(a, b, c, d, e, f, g, h, i)    i, h, g, f, e, d, c, b, a
#  define TO_ECC_640(a, b, c, d, e, f, g, h, i, j) j, i, h, g, f, e, d, c, b, a

#  define BN_MIN_ALLOC(bytes) \
      (BYTES_TO_CRYPT_WORDS(bytes) == 0) ? 1 : BYTES_TO_CRYPT_WORDS(bytes)
#  define ECC_CONST(NAME, bytes, initializer)                  \
      const struct                                             \
      {                                                        \
          crypt_uword_t allocate, size, d[BN_MIN_ALLOC(bytes)]; \
      } NAME = {BN_MIN_ALLOC(bytes), BYTES_TO_CRYPT_WORDS(bytes), {initializer}}

// This file contains the raw data for ECC curve constants. The data is wrapped
// in macros so this file can be included in other files that format the data in
// a memory format desired by the user.  This file itself is never used alone.
#  include <EccConstantData.inl>

// now define the TPMBN_ECC_CURVE_CONSTANTS objects for the known curves
```

```
#  if ECC_NIST_P192
const TPMBN_ECC_CURVE_CONSTANTS NIST_P192 = {TPM_ECC_NIST_P192,
                                             (bigNum)&NIST_P192_p,
                                             (bigNum)&NIST_P192_n,
                                             (bigNum)&NIST_P192_h,
                                             (bigNum)&NIST_P192_a,
                                             (bigNum)&NIST_P192_b,
                                             {(bigNum)&NIST_P192_gX,
                                              (bigNum)&NIST_P192_gY,
                                              (bigNum)&NIST_P192_gZ}};
#  endif  // ECC_NIST_P192

#  if ECC_NIST_P224
const TPMBN_ECC_CURVE_CONSTANTS NIST_P224 = {TPM_ECC_NIST_P224,
                                             (bigNum)&NIST_P224_p,
                                             (bigNum)&NIST_P224_n,
                                             (bigNum)&NIST_P224_h,
                                             (bigNum)&NIST_P224_a,
                                             (bigNum)&NIST_P224_b,
                                             {(bigNum)&NIST_P224_gX,
                                              (bigNum)&NIST_P224_gY,
                                              (bigNum)&NIST_P224_gZ}};
#  endif  // ECC_NIST_P224

#  if ECC_NIST_P256
const TPMBN_ECC_CURVE_CONSTANTS NIST_P256 = {TPM_ECC_NIST_P256,
                                             (bigNum)&NIST_P256_p,
                                             (bigNum)&NIST_P256_n,
                                             (bigNum)&NIST_P256_h,
                                             (bigNum)&NIST_P256_a,
                                             (bigNum)&NIST_P256_b,
                                             {(bigNum)&NIST_P256_gX,
                                              (bigNum)&NIST_P256_gY,
                                              (bigNum)&NIST_P256_gZ}};
#  endif  // ECC_NIST_P256

#  if ECC_NIST_P384
const TPMBN_ECC_CURVE_CONSTANTS NIST_P384 = {TPM_ECC_NIST_P384,
                                             (bigNum)&NIST_P384_p,
                                             (bigNum)&NIST_P384_n,
                                             (bigNum)&NIST_P384_h,
                                             (bigNum)&NIST_P384_a,
                                             (bigNum)&NIST_P384_b,
                                             {(bigNum)&NIST_P384_gX,
                                              (bigNum)&NIST_P384_gY,
                                              (bigNum)&NIST_P384_gZ}};
#  endif  // ECC_NIST_P384

#  if ECC_NIST_P521
const TPMBN_ECC_CURVE_CONSTANTS NIST_P521 = {TPM_ECC_NIST_P521,
                                             (bigNum)&NIST_P521_p,
                                             (bigNum)&NIST_P521_n,
                                             (bigNum)&NIST_P521_h,
                                             (bigNum)&NIST_P521_a,
                                             (bigNum)&NIST_P521_b,
                                             {(bigNum)&NIST_P521_gX,
                                              (bigNum)&NIST_P521_gY,
                                              (bigNum)&NIST_P521_gZ}};
#  endif  // ECC_NIST_P521

#  if ECC_BN_P256
const TPMBN_ECC_CURVE_CONSTANTS BN_P256 = {TPM_ECC_BN_P256,
                                           (bigNum)&BN_P256_p,
                                           (bigNum)&BN_P256_n,
                                           (bigNum)&BN_P256_h,
                                           (bigNum)&BN_P256_a,
```

```
                                                (bigNum)&BN_P256_b,
                                                {(bigNum)&BN_P256_gX,
                                                 (bigNum)&BN_P256_gY,
                                                 (bigNum)&BN_P256_gZ}};
#  endif  // ECC_BN_P256

#  if ECC_BN_P638
const TPMBN_ECC_CURVE_CONSTANTS BN_P638 = {TPM_ECC_BN_P638,
                                                (bigNum)&BN_P638_p,
                                                (bigNum)&BN_P638_n,
                                                (bigNum)&BN_P638_h,
                                                (bigNum)&BN_P638_a,
                                                (bigNum)&BN_P638_b,
                                                {(bigNum)&BN_P638_gX,
                                                 (bigNum)&BN_P638_gY,
                                                 (bigNum)&BN_P638_gZ}};
#  endif  // ECC_BN_P638

#  if ECC_SM2_P256
const TPMBN_ECC_CURVE_CONSTANTS SM2_P256 = {TPM_ECC_SM2_P256,
                                                 (bigNum)&SM2_P256_p,
                                                 (bigNum)&SM2_P256_n,
                                                 (bigNum)&SM2_P256_h,
                                                 (bigNum)&SM2_P256_a,
                                                 (bigNum)&SM2_P256_b,
                                                 {(bigNum)&SM2_P256_gX,
                                                  (bigNum)&SM2_P256_gY,
                                                  (bigNum)&SM2_P256_gZ}};
#  endif  // ECC_SM2_P256

#  define comma
const TPMBN_ECC_CURVE_CONSTANTS* bnEccCurveData[] = {
#  if ECC_NIST_P192
    &NIST_P192,
#  endif
#  if ECC_NIST_P224
    &NIST_P224,
#  endif
#  if ECC_NIST_P256
    &NIST_P256,
#  endif
#  if ECC_NIST_P384
    &NIST_P384,
#  endif
#  if ECC_NIST_P521
    &NIST_P521,
#  endif
#  if ECC_BN_P256
    &BN_P256,
#  endif
#  if ECC_BN_P638
    &BN_P638,
#  endif
#  if ECC_SM2_P256
    &SM2_P256,
#  endif
};

MUST_BE((sizeof(bnEccCurveData) / sizeof(bnEccCurveData[0])) == (ECC_CURVE_COUNT));

//*** BnGetCurveData()
// This function returns the pointer for the constant parameter data
// associated with a curve.
const TPMBN_ECC_CURVE_CONSTANTS* BnGetCurveData(TPM_ECC_CURVE curveId)
{
    for(int i = 0; i < ECC_CURVE_COUNT; i++)
```

```
    {
        if(bnEccCurveData[i]->curveId == curveId)
            return bnEccCurveData[i];
    }
    return NULL;
}

#endif  // TPM_ALG_ECC
```

/tpm/cryptolibs/TpmBigNum/BnMath.c

```
//** Introduction
// The simulator code uses the canonical form whenever possible in order to make
// the code in Part 3 more accessible. The canonical data formats are simple and
// not well suited for complex big number computations. When operating on big
// numbers, the data format is changed for easier manipulation. The format is native
// words in little-endian format. As the magnitude of the number decreases, the
// length of the array containing the number decreases but the starting address
// doesn't change.
//
// The functions in this file perform simple operations on these big numbers. Only
// the more complex operations are passed to the underlying support library.
// Although the support library would have most of these functions, the interface
// code to convert the format for the values is greater than the size of the
// code to implement the functions here. So, rather than incur the overhead of
// conversion, they are done here.
//
// If an implementer would prefer, the underlying library can be used simply by
// making code substitutions here.
//
// NOTE: There is an intention to continue to augment these functions so that there
// would be no need to use an external big number library.
//
// Many of these functions have no error returns and will always return TRUE. This
// is to allow them to be used in "guarded" sequences. That is:
//    OK = OK || BnSomething(s);
// where the BnSomething() function should not be called if OK isn't true.

//** Includes
#include "TpmBigNum.h"
extern BOOL g_inFailureMode;  // can't use global.h because we can't use tpm.h

// A constant value of zero as a stand in for NULL bigNum values
const bignum_t BnConstZero = {1, 0, {0}};

//** Functions

//*** AddSame()
// Adds two values that are the same size. This function allows 'result' to be
// the same as either of the addends. This is a nice function to put into assembly
// because handling the carry for multi-precision stuff is not as easy in C
// (unless there is a REALLY smart compiler). It would be nice if there were idioms
// in a language that a compiler could recognize what is going on and optimize
// loops like this.
//  Return Type: int
//      0           no carry out
//      1           carry out
static BOOL AddSame(crypt_uword_t*       result,
                    const crypt_uword_t* op1,
                    const crypt_uword_t* op2,
                    int                  count)
{
    int carry = 0;
    int i;
```

```
    for(i = 0; i < count; i++)
    {
        crypt_uword_t a   = op1[i];
        crypt_uword_t sum = a + op2[i];
        result[i]         = sum + carry;
        // generate a carry if the sum is less than either of the inputs
        // propagate a carry if there was a carry and the sum + carry is zero
        // do this using bit operations rather than logical operations so that
        // the time is about the same.
        //             propagate term      | generate term
        carry = ((result[i] == 0) & carry) | (sum < a);
    }
    return carry;
}

//*** CarryProp()
// Propagate a carry
static int CarryProp(
    crypt_uword_t* result, const crypt_uword_t* op, int count, int carry)
{
    for(; count; count--)
        carry = ((*result++ = *op++ + carry) == 0) & carry;
    return carry;
}

static void CarryResolve(bigNum result, int stop, int carry)
{
    if(carry)
    {
        pAssert((unsigned)stop < result->allocated);
        result->d[stop++] = 1;
    }
    BnSetTop(result, stop);
}

//*** BnAdd()
// This function adds two bigNum values. This function always returns TRUE.
LIB_EXPORT BOOL BnAdd(bigNum result, bigConst op1, bigConst op2)
{
    crypt_uword_t   stop;
    int             carry;
    const bignum_t* n1 = op1;
    const bignum_t* n2 = op2;

    //
    if(n2->size > n1->size)
    {
        n1 = op2;
        n2 = op1;
    }
    pAssert(result->allocated >= n1->size);
    stop  = MIN(n1->size, n2->allocated);
    carry = (int)AddSame(result->d, n1->d, n2->d, (int)stop);
    if(n1->size > stop)
        carry =
            CarryProp(&result->d[stop], &n1->d[stop], (int)(n1->size - stop), carry);
    CarryResolve(result, (int)n1->size, carry);
    return TRUE;
}

//*** BnAddWord()
// This function adds a word value to a bigNum. This function always returns TRUE.
LIB_EXPORT BOOL BnAddWord(bigNum result, bigConst op, crypt_uword_t word)
{
    int carry;
    //
```

```
    carry = (result->d[0] = op->d[0] + word) < word;
    carry = CarryProp(&result->d[1], &op->d[1], (int)(op->size - 1), carry);
    CarryResolve(result, (int)op->size, carry);
    return TRUE;
}

//*** SubSame()
// This function subtracts two values that have the same size.
static int SubSame(crypt_uword_t*       result,
                   const crypt_uword_t* op1,
                   const crypt_uword_t* op2,
                   int                  count)
{
    int borrow = 0;
    int i;
    for(i = 0; i < count; i++)
    {
        crypt_uword_t a    = op1[i];
        crypt_uword_t diff = a - op2[i];
        result[i]          = diff - borrow;
        //       generate   |      propagate
        borrow = (diff > a) | ((diff == 0) & borrow);
    }
    return borrow;
}

//*** BorrowProp()
// This propagates a borrow. If borrow is true when the end
// of the array is reached, then it means that op2 was larger than
// op1 and we don't handle that case so an assert is generated.
// This design choice was made because our only bigNum computations
// are on large positive numbers (primes) or on fields.
// Propagate a borrow.
static int BorrowProp(
    crypt_uword_t* result, const crypt_uword_t* op, int size, int borrow)
{
    for(; size > 0; size--)
        borrow = ((*result++ = *op++ - borrow) == MAX_CRYPT_UWORD) && borrow;
    return borrow;
}

//*** BnSub()
// This function does subtraction of two bigNum values and returns result = op1 - op2
// when op1 is greater than op2. If op2 is greater than op1, then a fault is
// generated. This function always returns TRUE.
LIB_EXPORT BOOL BnSub(bigNum result, bigConst op1, bigConst op2)
{
    int borrow;
    int stop = (int)MIN(op1->size, op2->allocated);
    //
    // Make sure that op2 is not obviously larger than op1
    pAssert(op1->size >= op2->size);
    borrow = SubSame(result->d, op1->d, op2->d, stop);
    if(op1->size > (crypt_uword_t)stop)
        borrow = BorrowProp(
            &result->d[stop], &op1->d[stop], (int)(op1->size - stop), borrow);
    pAssert(!borrow);
    BnSetTop(result, op1->size);
    return TRUE;
}

//*** BnSubWord()
// This function subtracts a word value from a bigNum. This function always
// returns TRUE.
LIB_EXPORT BOOL BnSubWord(bigNum result, bigConst op, crypt_uword_t word)
{
```

```
    int borrow;
    //
    pAssert(op->size > 1 || word <= op->d[0]);
    borrow       = word > op->d[0];
    result->d[0] = op->d[0] - word;
    borrow       = BorrowProp(&result->d[1], &op->d[1], (int)(op->size - 1), borrow);
    pAssert(!borrow);
    BnSetTop(result, op->size);
    return TRUE;
}

//*** BnUnsignedCmp()
// This function performs a comparison of op1 to op2. The compare is approximately
// constant time if the size of the values used in the compare is consistent
// across calls (from the same line in the calling code).
//  Return Type: int
//      < 0             op1 is less than op2
//      0               op1 is equal to op2
//      > 0             op1 is greater than op2
LIB_EXPORT int BnUnsignedCmp(bigConst op1, bigConst op2)
{
    int retVal;
    int diff;
    int i;
    //
    pAssert((op1 != NULL) && (op2 != NULL));
    retVal = (int)(op1->size - op2->size);
    if(retVal == 0)
    {
        for(i = (int)(op1->size - 1); i >= 0; i--)
        {
            diff   = (op1->d[i] < op2->d[i]) ? -1 : (op1->d[i] != op2->d[i]);
            retVal = retVal == 0 ? diff : retVal;
        }
    }
    else
        retVal = (retVal < 0) ? -1 : 1;
    return retVal;
}

//*** BnUnsignedCmpWord()
// Compare a bigNum to a crypt_uword_t.
//  Return Type: int
//      -1              op1 is less that word
//      0               op1 is equal to word
//      1               op1 is greater than word
LIB_EXPORT int BnUnsignedCmpWord(bigConst op1, crypt_uword_t word)
{
    if(op1->size > 1)
        return 1;
    else if(op1->size == 1)
        return (op1->d[0] < word) ? -1 : (op1->d[0] > word);
    else  // op1 is zero
        // equal if word is zero
        return (word == 0) ? 0 : -1;
}

//*** BnModWord()
// This function does modular division of a big number when the modulus is a
// word value.
LIB_EXPORT crypt_word_t BnModWord(bigConst numerator, crypt_word_t modulus)
{
    BN_MAX(remainder);
    BN_VAR(mod, RADIX_BITS);
    //
    mod->d[0] = modulus;
```

```
    mod->size = (modulus != 0);
    BnDiv(NULL, remainder, numerator, mod);
    return remainder->d[0];
}

//*** Msb()
// This function returns the bit number of the most significant bit of a
// crypt_uword_t. The number for the least significant bit of any bigNum value is 0.
// The maximum return value is RADIX_BITS - 1,
//  Return Type: int
//      -1              the word was zero
//      n               the bit number of the most significant bit in the word
static int Msb(crypt_uword_t word)
{
    int retVal = -1;
//
#if RADIX_BITS == 64
    if(word & 0xffffffff00000000)
    {
        retVal += 32;
        word >>= 32;
    }
#endif
    if(word & 0xffff0000)
    {
        retVal += 16;
        word >>= 16;
    }
    if(word & 0x0000ff00)
    {
        retVal += 8;
        word >>= 8;
    }
    if(word & 0x000000f0)
    {
        retVal += 4;
        word >>= 4;
    }
    if(word & 0x0000000c)
    {
        retVal += 2;
        word >>= 2;
    }
    if(word & 0x00000002)
    {
        retVal += 1;
        word >>= 1;
    }
    return retVal + (int)word;
}

//*** BnMsb()
// This function returns the number of the MSb of a bigNum value.
//  Return Type: int
//      -1              the word was zero or 'bn' was NULL
//      n               the bit number of the most significant bit in the word
LIB_EXPORT int BnMsb(bigConst bn)
{
    // If the value is NULL, or the size is zero then treat as zero and return -1
    if(bn != NULL && bn->size > 0)
    {
        int retVal = Msb(bn->d[bn->size - 1]);
        retVal += (int)(bn->size - 1) * RADIX_BITS;
        return retVal;
    }
    else
```

```
        return -1;
}

//*** BnSizeInBits()
// This function returns the number of bits required to hold a number. It is one
// greater than the Msb.
//
LIB_EXPORT unsigned BnSizeInBits(bigConst n)
{
    int bits = BnMsb(n) + 1;
    //
    return bits < 0 ? 0 : (unsigned)bits;
}

//*** BnSetWord()
// Change the value of a bignum_t to a word value.
LIB_EXPORT bigNum BnSetWord(bigNum n, crypt_uword_t w)
{
    if(n != NULL)
    {
        pAssert(n->allocated > 1);
        n->d[0] = w;
        BnSetTop(n, (w != 0) ? 1 : 0);
    }
    return n;
}

//*** BnSetBit()
// This function will SET a bit in a bigNum. Bit 0 is the least-significant bit in
// the 0th digit_t.  The function will return FALSE if the bitNum is invalid, else
TRUE.
LIB_EXPORT BOOL BnSetBit(bigNum       bn,     // IN/OUT: big number to modify
                         unsigned int bitNum  // IN: Bit number to SET
)
{
    crypt_uword_t offset = bitNum / RADIX_BITS;
    if(bitNum > bn->allocated * RADIX_BITS)
    {
        // out of range
        return FALSE;
    }
    // Grow the number if necessary to set the bit.
    while(bn->size <= offset)
        bn->d[bn->size++] = 0;
    bn->d[offset] |= ((crypt_uword_t)1 << RADIX_MOD(bitNum));
    return TRUE;
}

//*** BnTestBit()
// This function is used to check to see if a bit is SET in a bignum_t. The 0th bit
// is the LSb of d[0].
//  Return Type: BOOL
//      TRUE(1)         the bit is set
//      FALSE(0)        the bit is not set or the number is out of range
LIB_EXPORT BOOL BnTestBit(bigNum       bn,     // IN: number to check
                          unsigned int bitNum  // IN: bit to test
)
{
    crypt_uword_t offset = RADIX_DIV(bitNum);
    //
    if(bn->size > offset)
        return ((bn->d[offset] & (((crypt_uword_t)1) << RADIX_MOD(bitNum))) != 0);
    else
        return FALSE;
}
```

```
//***BnMaskBits()
// This function is used to mask off high order bits of a big number.
// The returned value will have no more than 'maskBit' bits
// set.
// Note: There is a requirement that unused words of a bignum_t are set to zero.
//  Return Type: BOOL
//      TRUE(1)         result masked
//      FALSE(0)        the input was not as large as the mask
LIB_EXPORT BOOL BnMaskBits(bigNum        bn,      // IN/OUT: number to mask
                           crypt_uword_t maskBit  // IN: the bit number for the mask.
)
{
    crypt_uword_t finalSize;
    BOOL          retVal;

    finalSize = BITS_TO_CRYPT_WORDS(maskBit);
    retVal    = (finalSize <= bn->allocated);
    if(retVal && (finalSize > 0))
    {
        crypt_uword_t mask;
        mask = ~((crypt_uword_t)0) >> RADIX_MOD(maskBit);
        bn->d[finalSize - 1] &= mask;
    }
    BnSetTop(bn, finalSize);
    return retVal;
}

//*** BnShiftRight()
// This function will shift a bigNum to the right by the shiftAmount.
// This function always returns TRUE.
LIB_EXPORT BOOL BnShiftRight(bigNum result, bigConst toShift, uint32_t shiftAmount)
{
    uint32_t      offset = (shiftAmount >> RADIX_LOG2);
    uint32_t      i;
    uint32_t      shiftIn;
    crypt_uword_t finalSize;
    //
    shiftAmount = shiftAmount & RADIX_MASK;
    shiftIn     = RADIX_BITS - shiftAmount;

    // The end size is toShift->size - offset less one additional
    // word if the shiftAmount would make the upper word == 0
    if(toShift->size > offset)
    {
        finalSize = toShift->size - offset;
        finalSize -= (toShift->d[toShift->size - 1] >> shiftAmount) == 0 ? 1 : 0;
    }
    else
        finalSize = 0;

    pAssert(finalSize <= result->allocated);
    if(finalSize != 0)
    {
        for(i = 0; i < finalSize; i++)
        {
            result->d[i] = (toShift->d[i + offset] >> shiftAmount)
                         | (toShift->d[i + offset + 1] << shiftIn);
        }
        if(offset == 0)
            result->d[i] = toShift->d[i] >> shiftAmount;
    }
    BnSetTop(result, finalSize);
    return TRUE;
}

//*** BnIsPointOnCurve()
```

```
// This function checks if a point is on the curve.
BOOL BnIsPointOnCurve(pointConst Q, const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    BN_VAR(right, (MAX_ECC_KEY_BITS * 3));
    BN_VAR(left, (MAX_ECC_KEY_BITS * 2));
    bigConst prime = BnCurveGetPrime(C);
    //
    // Show that point is on the curve y^2 = x^3 + ax + b;
    // Or y^2 = x(x^2 + a) + b
    // y^2
    BnMult(left, Q->y, Q->y);

    BnMod(left, prime);
    // x^2
    BnMult(right, Q->x, Q->x);

    // x^2 + a
    BnAdd(right, right, BnCurveGet_a(C));

    //    ExtMath_Mod(right, CurveGetPrime(C));
    // x(x^2 + a)
    BnMult(right, right, Q->x);

    // x(x^2 + a) + b
    BnAdd(right, right, BnCurveGet_b(C));

    BnMod(right, prime);
    if(BnUnsignedCmp(left, right) == 0)
        return TRUE;
    else
        return FALSE;
}
```

/tpm/cryptolibs/TpmBigNum/BnMemory.c

```
//** Introduction
// This file contains the memory setup functions used by the bigNum functions
// in CryptoEngine

//** Includes
#include "TpmBigNum.h"

//** Functions

//*** BnSetTop()
// This function is used when the size of a bignum_t is changed. It
// makes sure that the unused words are set to zero and that any significant
// words of zeros are eliminated from the used size indicator.
LIB_EXPORT bigNum BnSetTop(bigNum        bn,  // IN/OUT: number to clean
                           crypt_uword_t top  // IN: the new top
)
{
    if(bn != NULL)
    {
        pAssert(top <= bn->allocated);
        // If forcing the size to be decreased, make sure that the words being
        // discarded are being set to 0
        while(bn->size > top)
            bn->d[--bn->size] = 0;
        bn->size = top;
        // Now make sure that the words that are left are 'normalized' (no high-order
        // words of zero.
        while((bn->size > 0) && (bn->d[bn->size - 1] == 0))
            bn->size -= 1;
    }
```

```
    return bn;
}

//*** BnClearTop()
// This function will make sure that all unused words are zero.
LIB_EXPORT bigNum BnClearTop(bigNum bn)
{
    crypt_uword_t i;
    //
    if(bn != NULL)
    {
        for(i = bn->size; i < bn->allocated; i++)
            bn->d[i] = 0;
        while((bn->size > 0) && (bn->d[bn->size] == 0))
            bn->size -= 1;
    }
    return bn;
}

//*** BnInitializeWord()
// This function is used to initialize an allocated bigNum with a word value. The
// bigNum does not have to be allocated with a single word.
LIB_EXPORT bigNum BnInitializeWord(bigNum        bn,         // IN:
                                   crypt_uword_t allocated,  // IN:
                                   crypt_uword_t word        // IN:
)
{
    bn->allocated = allocated;
    bn->size      = (word != 0);
    bn->d[0]      = word;
    while(allocated > 1)
        bn->d[--allocated] = 0;
    return bn;
}

//*** BnInit()
// This function initializes a stack allocated bignum_t. It initializes
// 'allocated' and 'size' and zeros the words of 'd'.
LIB_EXPORT bigNum BnInit(bigNum bn, crypt_uword_t allocated)
{
    if(bn != NULL)
    {
        bn->allocated = allocated;
        bn->size      = 0;
        while(allocated != 0)
            bn->d[--allocated] = 0;
    }
    return bn;
}

//*** BnCopy()
// Function to copy a bignum_t. If the output is NULL, then
// nothing happens. If the input is NULL, the output is set
// to zero.
LIB_EXPORT BOOL BnCopy(bigNum out, bigConst in)
{
    if(in == out)
        BnSetTop(out, BnGetSize(out));
    else if(out != NULL)
    {
        if(in != NULL)
        {
            unsigned int i;
            pAssert(BnGetAllocated(out) >= BnGetSize(in));
            for(i = 0; i < BnGetSize(in); i++)
                out->d[i] = in->d[i];
```

```
            BnSetTop(out, BnGetSize(in));
        }
        else
            BnSetTop(out, 0);
    }
    return TRUE;
}

#if ALG_ECC

//*** BnPointCopy()
// Function to copy a bn point.
LIB_EXPORT BOOL BnPointCopy(bigPoint pOut, pointConst pIn)
{
    return BnCopy(pOut->x, pIn->x) && BnCopy(pOut->y, pIn->y)
           && BnCopy(pOut->z, pIn->z);
}

//*** BnInitializePoint()
// This function is used to initialize a point structure with the addresses
// of the coordinates.
LIB_EXPORT bn_point_t* BnInitializePoint(
    bigPoint p,  // OUT: structure to receive pointers
    bigNum   x,  // IN: x coordinate
    bigNum   y,  // IN: y coordinate
    bigNum   z   // IN: x coordinate
)
{
    p->x = x;
    p->y = y;
    p->z = z;
    BnSetWord(z, 1);
    return p;
}

#endif  // ALG_ECC
```

/tpm/cryptolibs/TpmBigNum/BnUtil.c

```
//** Introduction
// Utility functions to support TpmBigNum library

#include "TpmBigNum.h"
#include <CryptoInterface.h>

#if CRYPTO_LIB_REPORTING

//*** _crypto_GetMathImpl()
// Report the library being used for math.
void _crypto_GetMathImpl(_CRYPTO_IMPL_DESCRIPTION* result)
{
    // TpmmBigNum relies on a sub-library for its implementation.
    // Query the sub-library being used and use that to fill out the response.
    _CRYPTO_IMPL_DESCRIPTION subResult;
    BnGetImplementation(&subResult);

    // _CRYPTO_IMPL_DESCRIPTION has room for 31 characters plus NUL, and we use
    // 10 characters for the prefix "TPMBigNum/".
    // Using '%.21s' in snprintf below allows us to be safe and explicit about
    // the fact that we expect truncation of the name of the bignum sub-provider
    // in the event that its name is too long.
    snprintf(result->name, sizeof(result->name), "TPMBigNum/%.21s", subResult.name);
    snprintf(result->version, sizeof(result->version), "%s", subResult.version);
}
```

```
#endif  // CRYPTO_LIB_REPORTING
```

/tpm/cryptolibs/TpmBigNum/TpmBigNum.h

```
//** Introduction
// This file contains the headers necessary to build the tpm big num library.
// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/tpm_public.h>
#include <public/prototypes/TpmFail_fp.h>
// TODO_RENAME_INC_FOLDER: private refers to the TPM_CoreLib private(protected)
headers
#include <public/TpmAlgorithmDefines.h>
#include <public/GpMacros.h>  // required for TpmFail_fp.h
#include <public/Capabilities.h>
#include <public/TpmTypes.h>  // requires capabilities & GpMacros
#include <TpmBigNum/TpmToTpmBigNumMath.h>
#include "BnSupport_Interface.h"
#include "BnConvert_fp.h"
#include "BnMemory_fp.h"
#include "BnMath_fp.h"
#include "BnUtil_fp.h"
#include <MathLibraryInterface.h>
```

/tpm/cryptolibs/TpmBigNum/TpmBigNumThunks.c

```
//** Introduction
// This file contains BN Thunks between the MathInterfaceLibrary types and the
// bignum_t types.

#include "TpmBigNum.h"

// Note - these were moved out of TPM_INLINE to build correctly on GCC.  On MSVC
// link time code generation correctly handles the inline versions, but
// it isn't portable to GCC.

// ***************************************************************************
// Library Level Functions
// ***************************************************************************

// Called when system is initializing to allow math libraries to perform
// startup actions.
LIB_EXPORT int ExtMath_LibInit(void)
{
    return BnSupportLibInit();
}

//** MathLibraryCompatibililtyCheck()
// This function is only used during development to make sure that the library
// that is being referenced is using the same size of data structures as the TPM.
LIB_EXPORT BOOL ExtMath_Debug_CompatibilityCheck(void)
{
    return BnMathLibraryCompatibilityCheck();
}

// ***************************************************************************
// Integer/Number Functions (non-ECC)
// ***************************************************************************
// ##################
// type initializers
// ##################
LIB_EXPORT Crypt_Int* ExtMath_Initialize_Int(Crypt_Int* var, NUMBYTES bitCount)
{
    return (Crypt_Int*)BnInit((bigNum)var, BN_STRUCT_ALLOCATION(bitCount));
}
```

```
// ##################
// Buffer Converters
// ##################
LIB_EXPORT Crypt_Int* ExtMath_IntFromBytes(
    Crypt_Int* buffer, const BYTE* input, NUMBYTES byteCount)
{
    return (Crypt_Int*)BnFromBytes((bigNum)buffer, input, byteCount);
}

LIB_EXPORT BOOL ExtMath_IntToBytes(
    const Crypt_Int* value, BYTE* output, NUMBYTES* pByteCount)
{
    return BnToBytes((bigConst)value, output, pByteCount);
}

LIB_EXPORT Crypt_Int* ExtMath_SetWord(Crypt_Int* n, crypt_uword_t w)
{
    return (Crypt_Int*)BnSetWord((bigNum)n, w);
}
// ##################
// Copy Functions
// ##################
LIB_EXPORT BOOL ExtMath_Copy(Crypt_Int* out, const Crypt_Int* in)
{
    return BnCopy((bigNum)out, (bigConst)in);
}

// ################################
// Ordinary Arithmetic, writ large
// ################################

//** ExtMath_Multiply()
// Multiplies two numbers and returns the result
LIB_EXPORT BOOL ExtMath_Multiply(
    Crypt_Int* result, const Crypt_Int* multiplicand, const Crypt_Int* multiplier)
{
    return BnMult((bigNum)result, (bigConst)multiplicand, (bigConst)multiplier);
}

//** ExtMath_Divide()
// This function divides two Crypt_Int* values. The function returns FALSE if there is
// an error in the operation. Quotient may be null, in which case this function
returns
// only the remainder.
LIB_EXPORT BOOL ExtMath_Divide(Crypt_Int*       quotient,
                               Crypt_Int*       remainder,
                               const Crypt_Int* dividend,
                               const Crypt_Int* divisor)
{
    return BnDiv(
        (bigNum)quotient, (bigNum)remainder, (bigConst)dividend, (bigConst)divisor);
}

#if ALG_RSA
//** ExtMath_GCD()
// Get the greatest common divisor of two numbers. This function is only needed
// when the TPM implements RSA.
LIB_EXPORT BOOL ExtMath_GCD(
    Crypt_Int* gcd, const Crypt_Int* number1, const Crypt_Int* number2)
{
    return BnGcd((bigNum)gcd, (bigConst)number1, (bigConst)number2);
}
#endif  // ALG_RSA

//*** ExtMath_Add()
```

```
// This function adds two Crypt_Int* values. This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_Add(
    Crypt_Int* result, const Crypt_Int* op1, const Crypt_Int* op2)
{
    return BnAdd((bigNum)result, (bigConst)op1, (bigConst)op2);
}

//*** ExtMath_AddWord()
// This function adds a word value to a Crypt_Int*. This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_AddWord(
    Crypt_Int* result, const Crypt_Int* op, crypt_uword_t word)
{
    return BnAddWord((bigNum)result, (bigConst)op, word);
}

//*** ExtMath_Subtract()
// This function does subtraction of two Crypt_Int* values and returns result = op1 -
op2
// when op1 is greater than op2. If op2 is greater than op1, then a fault is
// generated. This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_Subtract(
    Crypt_Int* result, const Crypt_Int* op1, const Crypt_Int* op2)
{
    return BnSub((bigNum)result, (bigConst)op1, (bigConst)op2);
}

//*** ExtMath_SubtractWord()
// This function subtracts a word value from a Crypt_Int*. This function always
// returns TRUE.
LIB_EXPORT BOOL ExtMath_SubtractWord(
    Crypt_Int* result, const Crypt_Int* op, crypt_uword_t word)
{
    return BnSubWord((bigNum)result, (bigConst)op, word);
}

// ################################
// Modular Arithmetic, writ large
// ################################
// define Mod in terms of Divide
LIB_EXPORT BOOL ExtMath_Mod(Crypt_Int* valueAndResult, const Crypt_Int* modulus)
{
    return ExtMath_Divide(NULL, valueAndResult, valueAndResult, modulus);
}

//** ExtMath_ModMult()
// Does 'op1' * 'op2' and divide by 'modulus' returning the remainder of the divide.
LIB_EXPORT BOOL ExtMath_ModMult(Crypt_Int*       result,
                                const Crypt_Int* op1,
                                const Crypt_Int* op2,
                                const Crypt_Int* modulus)
{
    return BnModMult((bigNum)result, (bigConst)op1, (bigConst)op2, (bigConst)modulus);
}

#if ALG_RSA
//** ExtMath_ModExp()
// Do modular exponentiation using Crypt_Int* values. This function is only needed
// when the TPM implements RSA.
LIB_EXPORT BOOL ExtMath_ModExp(Crypt_Int*       result,
                               const Crypt_Int* number,
                               const Crypt_Int* exponent,
                               const Crypt_Int* modulus)
{
    return BnModExp(
        (bigNum)result, (bigConst)number, (bigConst)exponent, (bigConst)modulus);
}
```

```
#endif  // ALG_RSA

//** ExtMath_ModInverse()
// Modular multiplicative inverse.
LIB_EXPORT BOOL ExtMath_ModInverse(
    Crypt_Int* result, const Crypt_Int* number, const Crypt_Int* modulus)
{
    return BnModInverse((bigNum)result, (bigConst)number, (bigConst)modulus);
}

//*** ExtMath_ModWord()
// This function does modular division of a big number when the modulus is a
// word value.
LIB_EXPORT crypt_word_t ExtMath_ModWord(const Crypt_Int* numerator,
                                        crypt_word_t     modulus)
{
    return BnModWord((bigConst)numerator, modulus);
}

// ################################
// Queries
// ################################

//*** ExtMath_UnsignedCmp()
// This function performs a comparison of op1 to op2. The compare is approximately
// constant time if the size of the values used in the compare is consistent
// across calls (from the same line in the calling code).
//  Return Type: int
//      < 0             op1 is less than op2
//      0               op1 is equal to op2
//      > 0             op1 is greater than op2
LIB_EXPORT int ExtMath_UnsignedCmp(const Crypt_Int* op1, const Crypt_Int* op2)
{
    return BnUnsignedCmp((bigConst)op1, (bigConst)op2);
}

//*** ExtMath_UnsignedCmpWord()
// Compare a Crypt_Int* to a crypt_uword_t.
//  Return Type: int
//      -1              op1 is less that word
//      0               op1 is equal to word
//      1               op1 is greater than word
LIB_EXPORT int ExtMath_UnsignedCmpWord(const Crypt_Int* op1, crypt_uword_t word)
{
    return BnUnsignedCmpWord((bigConst)op1, word);
}

LIB_EXPORT BOOL ExtMath_IsEqualWord(const Crypt_Int* bn, crypt_uword_t word)
{
    return BnEqualWord((bigConst)bn, word);
}

LIB_EXPORT BOOL ExtMath_IsZero(const Crypt_Int* op1)
{
    return BnEqualZero((bigConst)op1);
}

//*** ExtMath_MostSigBitNum()
// This function returns the number of the MSb of a Crypt_Int* value.
//  Return Type: int
//      -1              the word was zero or 'bn' was NULL
//      n               the bit number of the most significant bit in the word
LIB_EXPORT int ExtMath_MostSigBitNum(const Crypt_Int* bn)
{
    return BnMsb((bigConst)bn);
}
```

```
LIB_EXPORT uint32_t ExtMath_GetLeastSignificant32bits(const Crypt_Int* bn)
{
    MUST_BE(RADIX_BITS >= 32);
#if RADIX_BITS == 32
    return BnGetWord(bn, 0);
#else
    // RADIX_BITS must be > 32 by MUST_BE above.
    return (uint32_t)(BnGetWord(bn, 0) & 0xFFFFFFFF);
#endif
}

//*** ExtMath_SizeInBits()
// This function returns the number of bits required to hold a number. It is one
// greater than the Msb.
LIB_EXPORT unsigned ExtMath_SizeInBits(const Crypt_Int* n)
{
    return BnSizeInBits((bigConst)n);
}

// ################################
// Bitwise Operations
// ################################

LIB_EXPORT BOOL ExtMath_SetBit(Crypt_Int* bn, unsigned int bitNum)
{
    return BnSetBit((bigNum)bn, bitNum);
}

// This function is used to check to see if a bit is SET in a bigNum_t. The 0th bit
//*** ExtMath_TestBit()
// is the LSb of d[0].
//  Return Type: BOOL
//      TRUE(1)         the bit is set
//      FALSE(0)        the bit is not set or the number is out of range
LIB_EXPORT BOOL ExtMath_TestBit(Crypt_Int*   bn,     // IN: number to check
                                unsigned int bitNum  // IN: bit to test
)
{
    return BnTestBit((bigNum)bn, bitNum);
}

//***ExtMath_MaskBits()
// This function is used to mask off high order bits of a big number.
// The returned value will have no more than 'maskBit' bits
// set.
// Note: There is a requirement that unused words of a bigNum_t are set to zero.
//  Return Type: BOOL
//      TRUE(1)         result masked
//      FALSE(0)        the input was not as large as the mask
LIB_EXPORT BOOL ExtMath_MaskBits(
    Crypt_Int*    bn,      // IN/OUT: number to mask
    crypt_uword_t maskBit  // IN: the bit number for the mask.
)
{
    return BnMaskBits((bigNum)bn, maskBit);
}

//*** ExtMath_ShiftRight()
// This function will shift a Crypt_Int* to the right by the shiftAmount.
// This function always returns TRUE.
LIB_EXPORT BOOL ExtMath_ShiftRight(
    Crypt_Int* result, const Crypt_Int* toShift, uint32_t shiftAmount)
{
    return BnShiftRight((bigNum)result, (bigConst)toShift, shiftAmount);
}
```

```
// ****************************************************************************
// ECC Functions
// ****************************************************************************
// ###################
// Point initializers
// ###################
LIB_EXPORT Crypt_Point* ExtEcc_Initialize_Point(Crypt_Point* point, NUMBYTES bitCount)
{
    // Since we define the structure, we know that BN_POINT_BUFs are a bn_point_t
followed by bignums.
    // and that the size is always the MAX_ECC_KEY_SIZE
    // tell the individual bignums how large they are:
    bn_fullpoint_t* pBuf = (bn_fullpoint_t*)point;
    BnInit((bigNum) & (pBuf->x), BN_STRUCT_ALLOCATION(bitCount));
    BnInit((bigNum) & (pBuf->y), BN_STRUCT_ALLOCATION(bitCount));
    BnInit((bigNum) & (pBuf->z), BN_STRUCT_ALLOCATION(bitCount));

    // now feed the addresses of those coordinates to the bn_point_t structure
    bn_point_t* bnPoint = (bn_point_t*)point;
    BnInitializePoint(
        bnPoint, (bigNum) & (pBuf->x), (bigNum) & (pBuf->y), (bigNum) & (pBuf->z));
    return point;
}

// ###################
// Curve initializers
// ###################
LIB_EXPORT const Crypt_EccCurve* ExtEcc_CurveInitialize(Crypt_EccCurve* E,
                                                        TPM_ECC_CURVE   curveId)
{
    return BnCurveInitialize((bigCurveData*)E, curveId);
}

// ##################
// Curve DESTRUCTOR
// ##################
// WARNING: Not guaranteed to be called in presence of LONGJMP.
LIB_EXPORT void ExtEcc_CurveFree(const Crypt_EccCurve* E)
{
    BnCurveFree((bigCurveData*)E);
}

// ##################
// Buffer Converters
// ##################
//*** BnPointFromBytes()
// Function to create a BIG_POINT structure from a 2B point.
// A point is going to be two ECC values in the same buffer. The values are going
// to be the size of the modulus.  They are in modular form.
LIB_EXPORT Crypt_Point* ExtEcc_PointFromBytes(Crypt_Point* point,
                                              const BYTE*  x,
                                              NUMBYTES     nBytesX,
                                              const BYTE*  y,
                                              NUMBYTES     nBytesY)
{
    return (Crypt_Point*)BnPointFromBytes((bigPoint)point, x, nBytesX, y, nBytesY);
}

LIB_EXPORT BOOL ExtEcc_PointToBytes(
    const Crypt_Point* point, BYTE* x, NUMBYTES* pBytesX, BYTE* y, NUMBYTES* pBytesY)
{
    return BnPointToBytes((pointConst)point, x, pBytesX, y, pBytesY);
}

// ####################
```

```
// ECC Point Operations
// ####################
//** ExtEcc_PointMultiply()
// This function does a point multiply of the form R = [d]S. A return of FALSE
// indicates that the result was the point at infinity. This function is only needed
// if the TPM supports ECC.
LIB_EXPORT BOOL ExtEcc_PointMultiply(
    Crypt_Point* R, const Crypt_Point* S, const Crypt_Int* d, const Crypt_EccCurve* E)
{
    return BnEccModMult((bigPoint)R, (pointConst)S, (bigConst)d, (bigCurveData*)E);
}

//** ExtEcc_PointMultiplyAndAdd()
// This function does a point multiply of the form R = [d]S + [u]Q. A return of
// FALSE indicates that the result was the point at infinity. This function is only
// needed if the TPM supports ECC.
LIB_EXPORT BOOL ExtEcc_PointMultiplyAndAdd(Crypt_Point*          R,
                                           const Crypt_Point*    S,
                                           const Crypt_Int*      d,
                                           const Crypt_Point*    Q,
                                           const Crypt_Int*      u,
                                           const Crypt_EccCurve* E)
{
    return BnEccModMult2((bigPoint)R,
                         (pointConst)S,
                         (bigConst)d,
                         (pointConst)Q,
                         (bigConst)u,
                         (bigCurveData*)E);
}

LIB_EXPORT BOOL ExtEcc_PointAdd(Crypt_Point*          R,
                                const Crypt_Point*    S,
                                const Crypt_Point*    Q,
                                const Crypt_EccCurve* E)
{
    return BnEccAdd((bigPoint)R, (pointConst)S, (pointConst)Q, (bigCurveData*)E);
}

// ####################
// ECC Point Information
// ####################
LIB_EXPORT BOOL ExtEcc_IsPointOnCurve(const Crypt_Point* Q, const Crypt_EccCurve* E)
{
    return BnIsPointOnCurve((pointConst)Q, AccessCurveConstants(E));
}

LIB_EXPORT const Crypt_Int* ExtEcc_PointX(const Crypt_Point* point)
{
    return (const Crypt_Int*)(((pointConst)point)->x);
}

LIB_EXPORT BOOL ExtEcc_IsInfinityPoint(const Crypt_Point* point)
{
    return BnEqualZero(((pointConst)point)->z);
}

// ####################
// ECC Curve Information
// ####################
LIB_EXPORT const Crypt_Int* ExtEcc_CurveGetPrime(TPM_ECC_CURVE curveId)
{
    return (const Crypt_Int*)BnCurveGetPrime(BnGetCurveData(curveId));
}

LIB_EXPORT const Crypt_Int* ExtEcc_CurveGetOrder(TPM_ECC_CURVE curveId)
```

```
{
    return (const Crypt_Int*)BnCurveGetOrder(BnGetCurveData(curveId));
}

LIB_EXPORT const Crypt_Int* ExtEcc_CurveGetCofactor(TPM_ECC_CURVE curveId)
{
    return (const Crypt_Int*)BnCurveGetCofactor(BnGetCurveData(curveId));
}

LIB_EXPORT const Crypt_Int* ExtEcc_CurveGet_a(TPM_ECC_CURVE curveId)
{
    return (const Crypt_Int*)BnCurveGet_a(BnGetCurveData(curveId));
}

LIB_EXPORT const Crypt_Int* ExtEcc_CurveGet_b(TPM_ECC_CURVE curveId)
{
    return (const Crypt_Int*)BnCurveGet_b(BnGetCurveData(curveId));
}

LIB_EXPORT const Crypt_Point* ExtEcc_CurveGetG(TPM_ECC_CURVE curveId)
{
    return (const Crypt_Point*)BnCurveGetG(BnGetCurveData(curveId));
}

LIB_EXPORT const Crypt_Int* ExtEcc_CurveGetGx(TPM_ECC_CURVE curveId)
{
    return (const Crypt_Int*)BnCurveGetGx(BnGetCurveData(curveId));
}

LIB_EXPORT const Crypt_Int* ExtEcc_CurveGetGy(TPM_ECC_CURVE curveId)
{
    return (const Crypt_Int*)BnCurveGetGy(BnGetCurveData(curveId));
}

LIB_EXPORT TPM_ECC_CURVE ExtEcc_CurveGetCurveId(const Crypt_EccCurve* E)
{
    return BnCurveGetCurveId(AccessCurveConstants(E));
}
```

/tpm/cryptolibs/TpmBigNum/include/BnConvert_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _BN_CONVERT_FP_H_
#define _BN_CONVERT_FP_H_

//*** BnFromBytes()
// This function will convert a big-endian byte array to the internal number
// format. If bn is NULL, then the output is NULL. If bytes is null or the
// required size is 0, then the output is set to zero
LIB_EXPORT bigNum BnFromBytes(bigNum bn, const BYTE* bytes, NUMBYTES nBytes);

//*** BnFrom2B()
// Convert an TPM2B to a BIG_NUM.
// If the input value does not exist, or the output does not exist, or the input
// will not fit into the output the function returns NULL
LIB_EXPORT bigNum BnFrom2B(bigNum       bn,  // OUT:
                           const TPM2B* a2B  // IN: number to convert
);

//*** BnToBytes()
// This function converts a BIG_NUM to a byte array. It converts the bigNum to a
```

```
// big-endian byte string and sets 'size' to the normalized value. If  'size' is an
// input 0, then the receiving buffer is guaranteed to be large enough for the result
// and the size will be set to the size required for bigNum (leading zeros
// suppressed).
//
// The conversion for a little-endian machine simply requires that all significant
// bytes of the bigNum be reversed. For a big-endian machine, rather than
// unpack each word individually, the bigNum is converted to little-endian words,
// copied, and then converted back to big-endian.
LIB_EXPORT BOOL BnToBytes(bigConst  bn,
                          BYTE*     buffer,
                          NUMBYTES* size  // This the number of bytes that are
                                          // available in the buffer. The result
                                          // should be this big.
);

//*** BnTo2B()
// Function to convert a BIG_NUM to TPM2B.
// The TPM2B size is set to the requested 'size' which may require padding.
// If 'size' is non-zero and less than required by the value in 'bn' then an error
// is returned. If 'size' is zero, then the TPM2B is assumed to be large enough
// for the data and a2b->size will be adjusted accordingly.
LIB_EXPORT BOOL BnTo2B(bigConst bn,   // IN:
                       TPM2B*   a2B,  // OUT:
                       NUMBYTES size  // IN: the desired size
);
#if ALG_ECC

//*** BnPointFromBytes()
// Function to create a BIG_POINT structure from a byte buffer in big-endian order.
// A point is going to be two ECC values in the same buffer. The values are going
// to be the size of the modulus.  They are in modular form.
LIB_EXPORT bn_point_t* BnPointFromBytes(
    bigPoint    ecP,  // OUT: the preallocated point structure
    const BYTE* x,
    NUMBYTES    nBytesX,
    const BYTE* y,
    NUMBYTES    nBytesY);

//*** BnPointToBytes()
// This function converts a BIG_POINT into a TPMS_ECC_POINT. A TPMS_ECC_POINT
// contains two TPM2B_ECC_PARAMETER values. The maximum size of the parameters
// is dependent on the maximum EC key size used in an implementation.
// The presumption is that the TPMS_ECC_POINT is large enough to hold 2 TPM2B
// values, each as large as a MAX_ECC_PARAMETER_BYTES
LIB_EXPORT BOOL BnPointToBytes(
    pointConst ecP,  // OUT: the preallocated point structure
    BYTE*      x,
    NUMBYTES*  pBytesX,
    BYTE*      y,
    NUMBYTES*  pBytesY);
#endif  // ALG_ECC

#endif  // _BN_CONVERT_FP_H_
```

/tpm/cryptolibs/TpmBigNum/include/BnMath_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Aug 30, 2019  Time: 02:11:54PM
 */

#ifndef _BN_MATH_FP_H_
#define _BN_MATH_FP_H_
```

```
//*** BnAdd()
// This function adds two bigNum values. This function always returns TRUE.
LIB_EXPORT BOOL BnAdd(bigNum result, bigConst op1, bigConst op2);

//*** BnAddWord()
// This function adds a word value to a bigNum. This function always returns TRUE.
LIB_EXPORT BOOL BnAddWord(bigNum result, bigConst op, crypt_uword_t word);

//*** BnSub()
// This function does subtraction of two bigNum values and returns result = op1 - op2
// when op1 is greater than op2. If op2 is greater than op1, then a fault is
// generated. This function always returns TRUE.
LIB_EXPORT BOOL BnSub(bigNum result, bigConst op1, bigConst op2);

//*** BnSubWord()
// This function subtracts a word value from a bigNum. This function always
// returns TRUE.
LIB_EXPORT BOOL BnSubWord(bigNum result, bigConst op, crypt_uword_t word);

//*** BnUnsignedCmp()
// This function performs a comparison of op1 to op2. The compare is approximately
// constant time if the size of the values used in the compare is consistent
// across calls (from the same line in the calling code).
//  Return Type: int
//      < 0             op1 is less than op2
//      0               op1 is equal to op2
//      > 0             op1 is greater than op2
LIB_EXPORT int BnUnsignedCmp(bigConst op1, bigConst op2);

//*** BnUnsignedCmpWord()
// Compare a bigNum to a crypt_uword_t.
//  Return Type: int
//      -1              op1 is less that word
//      0               op1 is equal to word
//      1               op1 is greater than word
LIB_EXPORT int BnUnsignedCmpWord(bigConst op1, crypt_uword_t word);

//*** BnModWord()
// This function does modular division of a big number when the modulus is a
// word value.
LIB_EXPORT crypt_word_t BnModWord(bigConst numerator, crypt_word_t modulus);

//*** BnMsb()
// This function returns the number of the MSb of a bigNum value.
//  Return Type: int
//      -1              the word was zero or 'bn' was NULL
//      n               the bit number of the most significant bit in the word
LIB_EXPORT int BnMsb(bigConst bn);

//*** BnSizeInBits()
// This function returns the number of bits required to hold a number. It is one
// greater than the Msb.
//
LIB_EXPORT unsigned BnSizeInBits(bigConst n);

//*** BnSetWord()
// Change the value of a bignum_t to a word value.
LIB_EXPORT bigNum BnSetWord(bigNum n, crypt_uword_t w);

//*** BnSetBit()
// This function will SET a bit in a bigNum. Bit 0 is the least-significant bit in
// the 0th digit_t. The function always return TRUE
LIB_EXPORT BOOL BnSetBit(bigNum       bn,     // IN/OUT: big number to modify
                         unsigned int bitNum  // IN: Bit number to SET
);
```

```
//*** BnTestBit()
// This function is used to check to see if a bit is SET in a bignum_t. The 0th bit
// is the LSb of d[0].
//  Return Type: BOOL
//      TRUE(1)         the bit is set
//      FALSE(0)        the bit is not set or the number is out of range
LIB_EXPORT BOOL BnTestBit(bigNum       bn,     // IN: number to check
                          unsigned int bitNum  // IN: bit to test
);

//***BnMaskBits()
// This function is used to mask off high order bits of a big number.
// The returned value will have no more than 'maskBit' bits
// set.
// Note: There is a requirement that unused words of a bignum_t are set to zero.
//  Return Type: BOOL
//      TRUE(1)         result masked
//      FALSE(0)        the input was not as large as the mask
LIB_EXPORT BOOL BnMaskBits(bigNum        bn,      // IN/OUT: number to mask
                           crypt_uword_t maskBit  // IN: the bit number for the mask.
);

//*** BnShiftRight()
// This function will shift a bigNum to the right by the shiftAmount.
// This function always returns TRUE.
LIB_EXPORT BOOL BnShiftRight(bigNum result, bigConst toShift, uint32_t shiftAmount);

//*** BnGetCurveData()
// This function returns the pointer for the parameter data
// associated with a curve.
const TPMBN_ECC_CURVE_CONSTANTS* BnGetCurveData(TPM_ECC_CURVE curveId);

//*** BnIsPointOnCurve()
// This function checks if a point is on the curve.
BOOL BnIsPointOnCurve(pointConst Q, const TPMBN_ECC_CURVE_CONSTANTS* C);

#endif  // _BN_MATH_FP_H_
```

/tpm/cryptolibs/TpmBigNum/include/BnMemory_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:18PM
 */

#ifndef _BN_MEMORY_FP_H_
#define _BN_MEMORY_FP_H_

//*** BnSetTop()
// This function is used when the size of a bignum_t is changed. It
// makes sure that the unused words are set to zero and that any significant
// words of zeros are eliminated from the used size indicator.
LIB_EXPORT bigNum BnSetTop(bigNum        bn,  // IN/OUT: number to clean
                           crypt_uword_t top  // IN: the new top
);

//*** BnClearTop()
// This function will make sure that all unused words are zero.
LIB_EXPORT bigNum BnClearTop(bigNum bn);

//*** BnInitializeWord()
// This function is used to initialize an allocated bigNum with a word value. The
// bigNum does not have to be allocated with a single word.
LIB_EXPORT bigNum BnInitializeWord(bigNum        bn,         // IN:
                                   crypt_uword_t allocated,  // IN:
```

```
                                    crypt_uword_t word        // IN:
);

//*** BnInit()
// This function initializes a stack allocated bignum_t. It initializes
// 'allocated' and 'size' and zeros the words of 'd'.
LIB_EXPORT bigNum BnInit(bigNum bn, crypt_uword_t allocated);

//*** BnCopy()
// Function to copy a bignum_t. If the output is NULL, then
// nothing happens. If the input is NULL, the output is set
// to zero.
LIB_EXPORT BOOL BnCopy(bigNum out, bigConst in);
#if ALG_ECC

//*** BnPointCopy()
// Function to copy a bn point.
LIB_EXPORT BOOL BnPointCopy(bigPoint pOut, pointConst pIn);

//*** BnInitializePoint()
// This function is used to initialize a point structure with the addresses
// of the coordinates.
LIB_EXPORT bn_point_t* BnInitializePoint(
    bigPoint p,  // OUT: structure to receive pointers
    bigNum   x,  // IN: x coordinate
    bigNum   y,  // IN: y coordinate
    bigNum   z   // IN: x coordinate
);
#endif  // ALG_ECC

#endif  // _BN_MEMORY_FP_H_
```

/tpm/cryptolibs/TpmBigNum/include/BnSupport_Interface.h

```
//** Introduction
// Prototypes for functions the bignum library requires
// from a bignum-based math support library.
// Functions contained in the MathInterface but not listed here are provided by
// the TpmBigNum library itself.
//
// This file contains the function prototypes for the functions that need to be
// present in the selected math library. For each function listed, there should
// be a small stub function. That stub provides the interface between the TPM
// code and the support library. In most cases, the stub function will only need
// to do a format conversion between the TPM big number and the support library
// big number. The TPM big number format was chosen to make this relatively
// simple and fast.
//
// Arithmetic operations return a BOOL to indicate if the operation completed
// successfully or not.

#ifndef BN_SUPPORT_INTERFACE_H
#define BN_SUPPORT_INTERFACE_H
// TODO_RENAME_INC_FOLDER:private refers to the TPM_CoreLib private headers
#include "public/GpMacros.h"
#include <CryptoInterface.h>
#include "BnValues.h"

//** BnSupportLibInit()
// This function is called by CryptInit() so that necessary initializations can be
// performed on the cryptographic library.
LIB_EXPORT
int BnSupportLibInit(void);

//** MathLibraryCompatibililtyCheck()
```

```
// This function is only used during development to make sure that the library
// that is being referenced is using the same size of data structures as the TPM.
BOOL BnMathLibraryCompatibilityCheck(void);

//** BnModMult()
// Does 'op1' * 'op2' and divide by 'modulus' returning the remainder of the divide.
LIB_EXPORT BOOL BnModMult(
    bigNum result, bigConst op1, bigConst op2, bigConst modulus);

//** BnMult()
// Multiplies two numbers and returns the result
LIB_EXPORT BOOL BnMult(bigNum result, bigConst multiplicand, bigConst multiplier);

//** BnDiv()
// This function divides two bigNum values. The function returns FALSE if there is
// an error in the operation.
LIB_EXPORT BOOL BnDiv(
    bigNum quotient, bigNum remainder, bigConst dividend, bigConst divisor);
//** BnMod()
#define BnMod(a, b) BnDiv(NULL, (a), (a), (b))

#if ALG_RSA
//** BnGcd()
// Get the greatest common divisor of two numbers. This function is only needed
// when the TPM implements RSA.
LIB_EXPORT BOOL BnGcd(bigNum gcd, bigConst number1, bigConst number2);

//** BnModExp()
// Do modular exponentiation using bigNum values. This function is only needed
// when the TPM implements RSA.
LIB_EXPORT BOOL BnModExp(
    bigNum result, bigConst number, bigConst exponent, bigConst modulus);
#endif  // ALG_RSA

//** BnModInverse()
// Modular multiplicative inverse.
LIB_EXPORT BOOL BnModInverse(bigNum result, bigConst number, bigConst modulus);

#if ALG_ECC

//** BnCurveInitialize()
// This function is used to initialize the pointers of a bigCurveData structure. The
// structure is a set of pointers to bigNum values. The curve-dependent values are
// set by a different function. This function is only needed
// if the TPM supports ECC.
LIB_EXPORT bigCurveData* BnCurveInitialize(bigCurveData* E, TPM_ECC_CURVE curveId);

//*** BnCurveFree()
// This function will free the allocated components of the curve and end the
// frame in which the curve data exists
LIB_EXPORT void BnCurveFree(bigCurveData* E);

//** BnEccModMult()
// This function does a point multiply of the form R = [d]S. A return of FALSE
// indicates that the result was the point at infinity. This function is only needed
// if the TPM supports ECC.
LIB_EXPORT BOOL BnEccModMult(
    bigPoint R, pointConst S, bigConst d, const bigCurveData* E);

//** BnEccModMult2()
// This function does a point multiply of the form R = [d]S + [u]Q. A return of
// FALSE indicates that the result was the point at infinity. This function is only
// needed if the TPM supports ECC.
LIB_EXPORT BOOL BnEccModMult2(bigPoint            R,
                              pointConst          S,
                              bigConst            d,
```

```
                                pointConst          Q,
                                bigConst            u,
                                const bigCurveData* E);

//** BnEccAdd()
// This function does a point add R = S + Q. A return of FALSE
// indicates that the result was the point at infinity. This function is only needed
// if the TPM supports ECC.
LIB_EXPORT BOOL BnEccAdd(
    bigPoint R, pointConst S, pointConst Q, const bigCurveData* E);

#endif  // ALG_ECC

#if CRYPTO_LIB_REPORTING

//** BnGetImplementation()
// This function reports the underlying library being used for bignum operations.
void BnGetImplementation(_CRYPTO_IMPL_DESCRIPTION* result);

#endif  // CRYPTO_LIB_REPORTING

#endif  //BN_SUPPORT_INTERFACE_H
```

/tpm/cryptolibs/TpmBigNum/include/BnUtil_fp.h

```
//** Introduction
// Utility functions to support TpmBigNum library
#ifndef _BNUTIL_FP_H_
#define _BNUTIL_FP_H_

#endif  // _BNUTIL_FP_H_
```

/tpm/cryptolibs/TpmBigNum/include/BnValues.h

```
//** Introduction

// This file contains the definitions needed for defining the internal bigNum
// structure.

// A bigNum is a pointer to a structure. The structure has three fields. The
// last field is and array (d) of crypt_uword_t. Each word is in machine format
// (big- or little-endian) with the words in ascending significance (i.e. words
// in little-endian order). This is the order that seems to be used in every
// big number library in the worlds, so...
//
// The first field in the structure (allocated) is the number of words in 'd'.
// This is the upper limit on the size of the number that can be held in the
// structure. This differs from libraries like OpenSSL as this is not intended
// to deal with numbers of arbitrary size; just numbers that are needed to deal
// with the algorithms that are defined in the TPM implementation.
//
// The second field in the structure (size) is the number of significant words
// in 'n'. When this number is zero, the number is zero. The word at used-1 should
// never be zero. All words between d[size] and d[allocated-1] should be zero.

//** Defines

#ifndef _BN_NUMBERS_H
#define _BN_NUMBERS_H
// TODO_RENAME_INC_FOLDER:private refers to the TPM_CoreLib private headers
#include <public/TpmAlgorithmDefines.h>
#include <public/GpMacros.h>  // required for TpmFail_fp.h
#include <public/Capabilities.h>
#include <public/TpmTypes.h>  // requires capabilities & GpMacros
```

```
// These are the basic big number formats. This is convertible to the library-
// specific format without too much difficulty. For the math performed using
// these numbers, the value is always positive.
#define BN_STRUCT_DEF(struct_type, count) \
    struct st_##struct_type##_t           \
    {                                     \
        crypt_uword_t allocated;          \
        crypt_uword_t size;               \
        crypt_uword_t d[count];           \
    }

typedef BN_STRUCT_DEF(bnroot, 1) bignum_t;

#ifndef bigNum
typedef bignum_t*       bigNum;
typedef const bignum_t* bigConst;
#endif  //bigNum

extern const bignum_t BnConstZero;

// The Functions to access the properties of a big number.
// Get number of allocated words
#define BnGetAllocated(x) (unsigned)((x)->allocated)

// Get number of words used
#define BnGetSize(x) ((x)->size)

// Get a pointer to the data array
#define BnGetArray(x) ((crypt_uword_t*)&((x)->d[0]))

// Get the nth word of a bigNum (zero-based)
#define BnGetWord(x, i) (crypt_uword_t)((x)->d[i])

// Some things that are done often.

// Test to see if a bignum_t is equal to zero
#define BnEqualZero(bn) (BnGetSize(bn) == 0)

// Test to see if a bignum_t is equal to a word type
#define BnEqualWord(bn, word) \
    ((BnGetSize(bn) == 1) && (BnGetWord(bn, 0) == (crypt_uword_t)word))

// Determine if a bigNum is even. A zero is even. Although the
// indication that a number is zero is that its size is zero,
// all words of the number are 0 so this test works on zero.
#define BnIsEven(n) ((BnGetWord(n, 0) & 1) == 0)

// The macros below are used to define bigNum values of the required
// size. The values are allocated on the stack so they can be
// treated like simple local values.

// This will call the initialization function for a defined bignum_t.
// This sets the allocated and used fields and clears the words of 'n'.
#define BN_INIT(name) \
    (bigNum) BnInit((bigNum) & (name), BYTES_TO_CRYPT_WORDS(sizeof(name.d)))

#define CRYPT_WORDS(bytes) BYTES_TO_CRYPT_WORDS(bytes)
#define MIN_ALLOC(bytes)   (CRYPT_WORDS(bytes) < 1 ? 1 : CRYPT_WORDS(bytes))
#define BN_CONST(name, bytes, initializer) \
    typedef const struct name##_type       \
    {                                      \
        crypt_uword_t allocated;           \
        crypt_uword_t size;                \
        crypt_uword_t d[MIN_ALLOC(bytes)]; \
    } name##_type;                         \
```

```
    name##_type name = {MIN_ALLOC(bytes), CRYPT_WORDS(bytes), {initializer}};

#define BN_STRUCT_ALLOCATION(bits) (BITS_TO_CRYPT_WORDS(bits) + 1)

// Create a structure of the correct size.
#define BN_STRUCT(struct_type, bits) \
    BN_STRUCT_DEF(struct_type, BN_STRUCT_ALLOCATION(bits))

// Define a bigNum type with a specific allocation
#define BN_TYPE(name, bits) typedef BN_STRUCT(name, bits) bn_##name##_t

// This creates a local bigNum variable of a specific size and
// initializes it from a TPM2B input parameter.
#define BN_INITIALIZED(name, bits, initializer) \
    BN_STRUCT(name, bits) name##_;              \
    bigNum name = TpmMath_IntFrom2B(BN_INIT(name##_), (const TPM2B*)initializer)

// Create a local variable that can hold a number with 'bits'
#define BN_VAR(name, bits)          \
    BN_STRUCT(name, bits) _##name; \
    bigNum name = BN_INIT(_##name)

// Create a type that can hold the largest number defined by the
// implementation.
#define BN_MAX(name) BN_VAR(name, LARGEST_NUMBER_BITS)
#define BN_MAX_INITIALIZED(name, initializer) \
    BN_INITIALIZED(name, LARGEST_NUMBER_BITS, initializer)

// A word size value is useful
#define BN_WORD(name) BN_VAR(name, RADIX_BITS)

// This is used to create a word-size bigNum and initialize it with
// an input parameter to a function.
#define BN_WORD_INITIALIZED(name, initial) \
    BN_STRUCT(RADIX_BITS) name##_;         \
    bigNum name = BnInitializeWord(        \
        (bigNum) & name##_, BN_STRUCT_ALLOCATION(RADIX_BITS), initial)

// ECC-Specific Values

// This is the format for a point. It is always in affine format. The Z value is
// carried as part of the point, primarily to simplify the interface to the support
// library. Rather than have the interface layer have to create space for the
// point each time it is used...
// The x, y, and z values are pointers to bigNum values and not in-line versions of
// the numbers. This is a relic of the days when there was no standard TPM format
// for the numbers
typedef struct _bn_point_t
{
    bigNum x;
    bigNum y;
    bigNum z;
} bn_point_t;

typedef bn_point_t*       bigPoint;
typedef const bn_point_t* pointConst;

typedef struct constant_point_t
{
    bigConst x;
    bigConst y;
    bigConst z;
} constant_point_t;

// coords points into x,y,z
// a bigPoint is a pointer to one of these structures, and
```

```
// therefore a pointer to bn_point_t (a coords).
// so bigPoint->coords->x->size is the size of x, and
// all 3 components are the same size.
#define BN_POINT_BUF(typename, bits)                \
    struct bnpt_st_##typename##_t                   \
    {                                               \
        bn_point_t coords;                          \
        BN_STRUCT(typename##_x, MAX_ECC_KEY_BITS) x; \
        BN_STRUCT(typename##_y, MAX_ECC_KEY_BITS) y; \
        BN_STRUCT(typename##_z, MAX_ECC_KEY_BITS) z; \
    }

typedef BN_POINT_BUF(fullpoint, MAX_ECC_KEY_BITS) bn_fullpoint_t;

// TPMBN_ECC_CURVE_CONSTANTS
// =========================
// A cryptographic elliptic curve is a mathematical set (Group) of points that
// satisfy the group equation and are generated by linear multiples of some
// initial "generator" point (Gx,Gy).
//
// The TPM code supports ECC Curves that satisfy equations of the following
// form:
//
// (y^2 = x^3 + a*x + b) mod p
//
// A particular cryptographic curve is fully described by the following
// parameters:
//
// | Name    | Meaning
|
// | :------ | :-----------------------------------------------------------------
------------ |
// | p       | curve prime
|
// | a, b    | equation coefficients
|
// | (Gx,Gy) | X and Y coordinates of the generator point.
|
// | n       | the order (size) of the generated group.  n must be prime.
|
// | h       | the cofactor of the group size to the full set of points for a
particular equation. |
//
// The group of constants to describe a particular ECC Curve (such as NIST P256
// or P384) are contained in TPMBN_ECC_CURVE_CONSTANTS objects.  In the
// TpmBigNum library these constants are always stored in TPM's internal BN
// (bigNum) format.
//
// Other math libraries are expected to provide these as compile time constants
// in a format they can efficiently consume at runtime.

// Structure for the curve parameters. This is an analog to the
// TPMS_ALGORITHM_DETAIL_ECC
typedef struct
{
    TPM_ECC_CURVE    curveId;  // TPM Algorithm ID for this data
    bigConst         prime;    // a prime number
    bigConst         order;    // the order of the curve
    bigConst         h;        // cofactor
    bigConst         a;        // linear coefficient
    bigConst         b;        // constant term
    constant_point_t base;     // base point
} TPMBN_ECC_CURVE_CONSTANTS;

// Access macros for the TPMBN_ECC_CURVE_CONSTANTS structure. The parameter 'C' is a
pointer
```

```
// to an TPMBN_ECC_CURVE_CONSTANTS structure. In some libraries, the curve structure E
contains
// a pointer to an TPMBN_ECC_CURVE_CONSTANTS structure as well as some other bits. For
those
// cases, the AccessCurveConstants function is used in the code to first get the
pointer
// to the TPMBN_ECC_CURVE_CONSTANTS for access. In some cases, the function does
nothing.
// AccessCurveConstants and these functions are all defined as inline so they can be
optimized
// away in cases where they are no-ops.
TPM_INLINE bigConst BnCurveGetPrime(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->prime;
}
TPM_INLINE bigConst BnCurveGetOrder(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->order;
}
TPM_INLINE bigConst BnCurveGetCofactor(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->h;
}
TPM_INLINE bigConst BnCurveGet_a(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->a;
}
TPM_INLINE bigConst BnCurveGet_b(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->b;
}
TPM_INLINE pointConst BnCurveGetG(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return (pointConst) & (C->base);
}
TPM_INLINE bigConst BnCurveGetGx(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->base.x;
}
TPM_INLINE bigConst BnCurveGetGy(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->base.y;
}
TPM_INLINE TPM_ECC_CURVE BnCurveGetCurveId(const TPMBN_ECC_CURVE_CONSTANTS* C)
{
    return C->curveId;
}

// Convert bytes in initializers
// This is used for CryptEccData.c.
#define BIG_ENDIAN_BYTES_TO_UINT32(a, b, c, d) \
    (((UINT32)(a) << 24) + ((UINT32)(b) << 16) + ((UINT32)(c) << 8) + ((UINT32)(d)))

#define BIG_ENDIAN_BYTES_TO_UINT64(a, b, c, d, e, f, g, h)             \
    (((UINT64)(a) << 56) + ((UINT64)(b) << 48) + ((UINT64)(c) << 40)   \
     + ((UINT64)(d) << 32) + ((UINT64)(e) << 24) + ((UINT64)(f) << 16) \
     + ((UINT64)(g) << 8) + ((UINT64)(h)))

// These macros are used for data initialization of big number ECC constants
// These two macros combine a macro for data definition with a macro for
// structure initialization. The 'a' parameter is a macro that gives numbers to
// each of the bytes of the initializer and defines where each of the numberd
// bytes will show up in the final structure. The 'b' value is a structure that
// contains the requisite number of bytes in big endian order. S, the MJOIN
// and JOIND macros will combine a macro defining a data layout with a macro defining
// the data to be places. Generally, these macros will only need expansion when
```

```
// CryptEccData.c gets compiled.
#define JOINED(a, b) a b
#define MJOIN(a, b)  a b

#if RADIX_BYTES == 64
#  define B8_TO_BN(a, b, c, d, e, f, g, h)                                    \
      ((((((((((((((((UINT64)a) << 8) | (UINT64)b) << 8) | (UINT64)c) << 8) \
              | (UINT64)d)                                                    \
             << 8)                                                            \
            | (UINT64)e)                                                      \
           << 8)                                                              \
          | (UINT64)f)                                                        \
         << 8)                                                                \
        | (UINT64)g)                                                          \
       << 8)                                                                  \
      | (UINT64)h)
#  define B1_TO_BN(a)                  B8_TO_BN(0, 0, 0, 0, 0, 0, 0, a)
#  define B2_TO_BN(a, b)               B8_TO_BN(0, 0, 0, 0, 0, 0, a, b)
#  define B3_TO_BN(a, b, c)            B8_TO_BN(0, 0, 0, 0, 0, a, b, c)
#  define B4_TO_BN(a, b, c, d)         B8_TO_BN(0, 0, 0, 0, a, b, c, d)
#  define B5_TO_BN(a, b, c, d, e)      B8_TO_BN(0, 0, 0, a, b, c, d, e)
#  define B6_TO_BN(a, b, c, d, e, f)   B8_TO_BN(0, 0, a, b, c, d, e, f)
#  define B7_TO_BN(a, b, c, d, e, f, g) B8_TO_BN(0, a, b, c, d, e, f, g)
#else
#  define B1_TO_BN(a)       B4_TO_BN(0, 0, 0, a)
#  define B2_TO_BN(a, b)    B4_TO_BN(0, 0, a, b)
#  define B3_TO_BN(a, b, c) B4_TO_BN(0, a, b, c)
#  define B4_TO_BN(a, b, c, d) \
      ((((((UINT32)a << 8) | (UINT32)b) << 8) | (UINT32)c) << 8) | (UINT32)d)
#  define B5_TO_BN(a, b, c, d, e)          B4_TO_BN(b, c, d, e), B1_TO_BN(a)
#  define B6_TO_BN(a, b, c, d, e, f)       B4_TO_BN(c, d, e, f), B2_TO_BN(a, b)
#  define B7_TO_BN(a, b, c, d, e, f, g)    B4_TO_BN(d, e, f, g), B3_TO_BN(a, b, c)
#  define B8_TO_BN(a, b, c, d, e, f, g, h) B4_TO_BN(e, f, g, h), B4_TO_BN(a, b, c, d)

#endif

#endif  // _BN_NUMBERS_H
```

/tpm/cryptolibs/TpmBigNum/include/TpmBigNum/TpmToTpmBigNumMath.h

```
//** Introduction
// This file contains OpenSSL specific functions called by TpmBigNum library to
provide
// the TpmBigNum + OpenSSL math support.

#ifndef _TPM_TO_TPMBIGNUM_MATH_H_
#define _TPM_TO_TPMBIGNUM_MATH_H_

#ifdef MATH_LIB_DEFINED
#  error only one primary math library allowed
#endif
#define MATH_LIB_DEFINED

// indicate the TPMBIGNUM library is active
#define MATH_LIB_TPMBIGNUM

// TODO_RENAME_INC_FOLDER: private refers to the TPM_CoreLib private headers
#include <public/GpMacros.h>  // required for TpmFail_fp.h
#include <public/Capabilities.h>
#include <public/TpmTypes.h>  // requires capabilities & GpMacros
#include "BnValues.h"

#ifndef LIB_INCLUDE
#  error include ordering error, LIB_INCLUDE not defined
#endif
```

```
#ifndef BN_MATH_LIB
#  error BN_MATH_LIB not defined, required to provide BN library functions.
#endif

#if defined(CRYPT_CURVE_INITIALIZED) || defined(CRYPT_CURVE_FREE)
#error include ordering error, expected CRYPT_CURVE_INITIALIZED & CRYPT_CURVE_FREE to
be undefined.
#endif

// Add support library dependent definitions.
// For TpmBigNum, we expect bigCurveData to be a defined type.
#include LIB_INCLUDE(BnTo, BN_MATH_LIB, Math)

#include "BnConvert_fp.h"
#include "BnMath_fp.h"
#include "BnMemory_fp.h"
#include "BnSupport_Interface.h"

// Define macros and types necessary for the math library abstraction layer
// Create a data object backing a Crypt_Int big enough for the given number of
// data bits
#define CRYPT_INT_BUF(buftypename, bits) BN_STRUCT(buftypename, bits)

// Create a data object backing a Crypt_Point big enough for the given number of
// data bits, per coordinate
#define CRYPT_POINT_BUF(buftypename, bits) BN_POINT_BUF(buftypename, bits)

// Create an instance of a data object underlying Crypt_EccCurve on the stack
// sufficient for given bit size.  In our case, all are the same size.
#define CRYPT_CURVE_BUF(buftypename, max_size_in_bits) bigCurveData

// now include the math library functional interface and instantiate the
// Crypt_Int & related types
// TODO_RENAME_INC_FOLDER: This should have a Tpm_Cryptolib_Common component prefix.
#include <MathLibraryInterface.h>

#endif  // _TPM_TO_TPMBIGNUM_MATH_H_
```

B.4 OpenSSL-Specific Files

Introduction

The following files are specific to a port that uses the OpenSSL library for cryptographic functions.

/tpm/cryptolibs/Ossl/BnToOsslMath.c

```
//** Introduction
// The functions in this file provide the low-level interface between the TPM code
// and the big number and elliptic curve math routines in OpenSSL.
//
// Most math on big numbers require a context. The context contains the memory in
// which OpenSSL creates and manages the big number values. When a OpenSSL math
// function will be called that modifies a BIGNUM value, that value must be created in
// an OpenSSL context. The first line of code in such a function must be:
// OSSL_ENTER(); and the last operation before returning must be OSSL_LEAVE().
// OpenSSL variables can then be created with BnNewVariable(). Constant values to be
// used by OpenSSL are created from the bigNum values passed to the functions in this
// file. Space for the BIGNUM control block is allocated in the stack of the
// function and then it is initialized by calling BigInitialized(). That function
// sets up the values in the BIGNUM structure and sets the data pointer to point to
// the data in the bignum_t. This is only used when the value is known to be a
// constant in the called function.
```

```
//
// Because the allocations of constants is on the local stack and the
// OSSL_ENTER()/OSSL_LEAVE() pair flushes everything created in OpenSSL memory, there
// should be no chance of a memory leak.

//** Includes and Defines
//#include "Tpm.h"
#include "BnOssl.h"

#ifdef MATH_LIB_OSSL
#  include <Ossl/BnToOsslMath_fp.h>

//** Functions

//*** OsslToTpmBn()
// This function converts an OpenSSL BIGNUM to a TPM bigNum. In this implementation
// it is assumed that OpenSSL uses a different control structure but the same data
// layout -- an array of native-endian words in little-endian order.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure because value will not fit or OpenSSL variable doesn't
//                      exist
BOOL OsslToTpmBn(bigNum bn, BIGNUM* osslBn)
{
    GOTO_ERROR_UNLESS(osslBn != NULL);
    // If the bn is NULL, it means that an output value pointer was NULL meaning that
    // the results is simply to be discarded.
    if(bn != NULL)
    {
        int i;
        //
        GOTO_ERROR_UNLESS((unsigned)osslBn->top <= BnGetAllocated(bn));
        for(i = 0; i < osslBn->top; i++)
            bn->d[i] = osslBn->d[i];
        BnSetTop(bn, osslBn->top);
    }
    return TRUE;
Error:
    return FALSE;
}

//*** BigInitialized()
// This function initializes an OSSL BIGNUM from a TPM bigConst. Do not use this for
// values that are passed to OpenSLL when they are not declared as const in the
// function prototype. Instead, use BnNewVariable().
BIGNUM* BigInitialized(BIGNUM* toInit, bigConst initializer)
{
    if(initializer == NULL)
        FAIL(FATAL_ERROR_PARAMETER);
    if(toInit == NULL || initializer == NULL)
        return NULL;
    toInit->d     = (BN_ULONG*)&initializer->d[0];
    toInit->dmax  = (int)initializer->allocated;
    toInit->top   = (int)initializer->size;
    toInit->neg   = 0;
    toInit->flags = 0;
    return toInit;
}

#  ifndef OSSL_DEBUG
#    define BIGNUM_PRINT(label, bn, eol)
#    define DEBUG_PRINT(x)
#  else
#    define DEBUG_PRINT(x)                printf("%s", x)
#    define BIGNUM_PRINT(label, bn, eol) BIGNUM_print((label), (bn), (eol))
```

```
//*** BIGNUM_print()
static void BIGNUM_print(const char* label, const BIGNUM* a, BOOL eol)
{
    BN_ULONG* d;
    int       i;
    int       notZero = FALSE;

    if(label != NULL)
        printf("%s", label);
    if(a == NULL)
    {
        printf("NULL");
        goto done;
    }
    if(a->neg)
        printf("-");
    for(i = a->top, d = &a->d[i - 1]; i > 0; i--)
    {
        int      j;
        BN_ULONG l = *d--;
        for(j = BN_BITS2 - 8; j >= 0; j -= 8)
        {
            BYTE b  = (BYTE)((l >> j) & 0xFF);
            notZero = notZero || (b != 0);
            if(notZero)
                printf("%02x", b);
        }
        if(!notZero)
            printf("0");
    }
done:
    if(eol)
        printf("\n");
    return;
}
#  endif

//*** BnNewVariable()
// This function allocates a new variable in the provided context. If the context
// does not exist or the allocation fails, it is a catastrophic failure.
static BIGNUM* BnNewVariable(BN_CTX* CTX)
{
    BIGNUM* new;
    //
    // This check is intended to protect against calling this function without
    // having initialized the CTX.
    if((CTX == NULL) || ((new = BN_CTX_get(CTX)) == NULL))
        FAIL(FATAL_ERROR_ALLOCATION);
    return new;
}

#  if LIBRARY_COMPATIBILITY_CHECK

//*** MathLibraryCompatibilityCheck()
BOOL BnMathLibraryCompatibilityCheck(void)
{
    OSSL_ENTER();
    BIGNUM*       osslTemp = BnNewVariable(CTX);
    crypt_uword_t i;
    BYTE test[] = {0x1F, 0x1E, 0x1D, 0x1C, 0x1B, 0x1A, 0x19, 0x18, 0x17, 0x16, 0x15,
                   0x14, 0x13, 0x12, 0x11, 0x10, 0x0F, 0x0E, 0x0D, 0x0C, 0x0B, 0x0A,
                   0x09, 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00};
    BN_VAR(tpmTemp, sizeof(test) * 8);  // allocate some space for a test value
                                        //
    // Convert the test data to a bigNum
    BnFromBytes(tpmTemp, test, sizeof(test));
```

```
    // Convert the test data to an OpenSSL BIGNUM
    BN_bin2bn(test, sizeof(test), osslTemp);
    // Make sure the values are consistent
    GOTO_ERROR_UNLESS(osslTemp->top == (int)tpmTemp->size);
    for(i = 0; i < tpmTemp->size; i++)
        GOTO_ERROR_UNLESS(osslTemp->d[i] == tpmTemp->d[i]);
    OSSL_LEAVE();
    return 1;
Error:
    return 0;
}
#  endif

//*** BnModMult()
// This function does a modular multiply. It first does a multiply and then a divide
// and returns the remainder of the divide.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation
LIB_EXPORT BOOL BnModMult(bigNum result, bigConst op1, bigConst op2, bigConst modulus)
{
    OSSL_ENTER();
    BOOL    OK       = TRUE;
    BIGNUM* bnResult = BN_NEW();
    BIGNUM* bnTemp   = BN_NEW();
    BIG_INITIALIZED(bnOp1, op1);
    BIG_INITIALIZED(bnOp2, op2);
    BIG_INITIALIZED(bnMod, modulus);
    //
    GOTO_ERROR_UNLESS(BN_mul(bnTemp, bnOp1, bnOp2, CTX));
    GOTO_ERROR_UNLESS(BN_div(NULL, bnResult, bnTemp, bnMod, CTX));
    GOTO_ERROR_UNLESS(OsslToTpmBn(result, bnResult));
    goto Exit;
Error:
    OK = FALSE;
Exit:
    OSSL_LEAVE();
    return OK;
}

//*** BnMult()
// Multiplies two numbers
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation
LIB_EXPORT BOOL BnMult(bigNum result, bigConst multiplicand, bigConst multiplier)
{
    OSSL_ENTER();
    BIGNUM* bnTemp = BN_NEW();
    BOOL    OK     = TRUE;
    BIG_INITIALIZED(bnA, multiplicand);
    BIG_INITIALIZED(bnB, multiplier);
    //
    GOTO_ERROR_UNLESS(BN_mul(bnTemp, bnA, bnB, CTX));
    GOTO_ERROR_UNLESS(OsslToTpmBn(result, bnTemp));
    goto Exit;
Error:
    OK = FALSE;
Exit:
    OSSL_LEAVE();
    return OK;
}

//*** BnDiv()
// This function divides two bigNum values. The function returns FALSE if
// there is an error in the operation.
```

```
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation
LIB_EXPORT BOOL BnDiv(
    bigNum quotient, bigNum remainder, bigConst dividend, bigConst divisor)
{
    OSSL_ENTER();
    BIGNUM* bnQ = BN_NEW();
    BIGNUM* bnR = BN_NEW();
    BOOL    OK  = TRUE;
    BIG_INITIALIZED(bnDend, dividend);
    BIG_INITIALIZED(bnSor, divisor);
    //
    if(BnEqualZero(divisor))
        FAIL(FATAL_ERROR_DIVIDE_ZERO);
    GOTO_ERROR_UNLESS(BN_div(bnQ, bnR, bnDend, bnSor, CTX));
    GOTO_ERROR_UNLESS(OsslToTpmBn(quotient, bnQ));
    GOTO_ERROR_UNLESS(OsslToTpmBn(remainder, bnR));
    DEBUG_PRINT("In BnDiv:\n");
    BIGNUM_PRINT("   bnDividend: ", bnDend, TRUE);
    BIGNUM_PRINT("    bnDivisor: ", bnSor, TRUE);
    BIGNUM_PRINT("   bnQuotient: ", bnQ, TRUE);
    BIGNUM_PRINT("  bnRemainder: ", bnR, TRUE);
    goto Exit;
Error:
    OK = FALSE;
Exit:
    OSSL_LEAVE();
    return OK;
}

#  if ALG_RSA
//*** BnGcd()
// Get the greatest common divisor of two numbers
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation
LIB_EXPORT BOOL BnGcd(bigNum   gcd,      // OUT: the common divisor
                      bigConst number1,  // IN:
                      bigConst number2   // IN:
)
{
    OSSL_ENTER();
    BIGNUM* bnGcd = BN_NEW();
    BOOL    OK    = TRUE;
    BIG_INITIALIZED(bn1, number1);
    BIG_INITIALIZED(bn2, number2);
    //
    GOTO_ERROR_UNLESS(BN_gcd(bnGcd, bn1, bn2, CTX));
    GOTO_ERROR_UNLESS(OsslToTpmBn(gcd, bnGcd));
    goto Exit;
Error:
    OK = FALSE;
Exit:
    OSSL_LEAVE();
    return OK;
}

//***BnModExp()
// Do modular exponentiation using bigNum values. The conversion from a bignum_t to
// a bigNum is trivial as they are based on the same structure
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation
LIB_EXPORT BOOL BnModExp(bigNum   result,    // OUT: the result
                         bigConst number,    // IN: number to exponentiate
```

```
                        bigConst exponent,  // IN:
                        bigConst modulus    // IN:
)
{
    OSSL_ENTER();
    BIGNUM* bnResult = BN_NEW();
    BOOL    OK       = TRUE;
    BIG_INITIALIZED(bnN, number);
    BIG_INITIALIZED(bnE, exponent);
    BIG_INITIALIZED(bnM, modulus);
    //
    GOTO_ERROR_UNLESS(BN_mod_exp(bnResult, bnN, bnE, bnM, CTX));
    GOTO_ERROR_UNLESS(OsslToTpmBn(result, bnResult));
    goto Exit;
Error:
    OK = FALSE;
Exit:
    OSSL_LEAVE();
    return OK;
}
#  endif  // ALG_RSA

//*** BnModInverse()
// Modular multiplicative inverse
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation
LIB_EXPORT BOOL BnModInverse(bigNum result, bigConst number, bigConst modulus)
{
    OSSL_ENTER();
    BIGNUM* bnResult = BN_NEW();
    BOOL    OK       = TRUE;
    BIG_INITIALIZED(bnN, number);
    BIG_INITIALIZED(bnM, modulus);
    //
    GOTO_ERROR_UNLESS(BN_mod_inverse(bnResult, bnN, bnM, CTX) != NULL);
    GOTO_ERROR_UNLESS(OsslToTpmBn(result, bnResult));
    goto Exit;
Error:
    OK = FALSE;
Exit:
    OSSL_LEAVE();
    return OK;
}

#  if ALG_ECC

//*** PointFromOssl()
// Function to copy the point result from an OSSL function to a bigNum
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation
static BOOL PointFromOssl(bigPoint            pOut,  // OUT: resulting point
                          EC_POINT*           pIn,   // IN: the point to return
                          const bigCurveData* E      // IN: the curve
)
{
    BIGNUM* x = NULL;
    BIGNUM* y = NULL;
    BOOL    OK;
    BN_CTX_start(E->CTX);
    //
    x = BN_CTX_get(E->CTX);
    y = BN_CTX_get(E->CTX);

    if(y == NULL)
```

```
        FAIL(FATAL_ERROR_ALLOCATION);
    // If this returns false, then the point is at infinity
    OK = EC_POINT_get_affine_coordinates_GFp(E->G, pIn, x, y, E->CTX);
    if(OK)
    {
        OsslToTpmBn(pOut->x, x);
        OsslToTpmBn(pOut->y, y);
        BnSetWord(pOut->z, 1);
    }
    else
        BnSetWord(pOut->z, 0);
    BN_CTX_end(E->CTX);
    return OK;
}

//*** EcPointInitialized()
// Allocate and initialize a point.
static EC_POINT* EcPointInitialized(pointConst initializer, const bigCurveData* E)
{
    EC_POINT* P = NULL;

    if(initializer != NULL)
    {
        BIG_INITIALIZED(bnX, initializer->x);
        BIG_INITIALIZED(bnY, initializer->y);
        if(E == NULL)
            FAIL(FATAL_ERROR_ALLOCATION);
        P = EC_POINT_new(E->G);
        if(!EC_POINT_set_affine_coordinates_GFp(E->G, P, bnX, bnY, E->CTX))
            P = NULL;
    }
    return P;
}

//*** BnCurveInitialize()
// This function initializes the OpenSSL curve information structure. This
// structure points to the TPM-defined values for the curve, to the context for the
// number values in the frame, and to the OpenSSL-defined group values.
//  Return Type: bigCurveData*
//      NULL        the TPM_ECC_CURVE is not valid or there was a problem in
//                  in initializing the curve data
//      non-NULL    points to 'E'
LIB_EXPORT bigCurveData* BnCurveInitialize(
    bigCurveData* E,       // IN: curve structure to initialize
    TPM_ECC_CURVE curveId  // IN: curve identifier
)
{
    const TPMBN_ECC_CURVE_CONSTANTS* C = BnGetCurveData(curveId);
    if(C == NULL)
        E = NULL;
    if(E != NULL)
    {
        // This creates the OpenSSL memory context that stays in effect as long as the
        // curve (E) is defined.
        OSSL_ENTER();  // if the allocation fails, the TPM fails
        EC_POINT* P = NULL;
        BIG_INITIALIZED(bnP, C->prime);
        BIG_INITIALIZED(bnA, C->a);
        BIG_INITIALIZED(bnB, C->b);
        BIG_INITIALIZED(bnX, C->base.x);
        BIG_INITIALIZED(bnY, C->base.y);
        BIG_INITIALIZED(bnN, C->order);
        BIG_INITIALIZED(bnH, C->h);
        //
        E->C   = C;
        E->CTX = CTX;
```

```
        // initialize EC group, associate a generator point and initialize the point
        // from the parameter data
        // Create a group structure
        E->G = EC_GROUP_new_curve_GFp(bnP, bnA, bnB, CTX);
        GOTO_ERROR_UNLESS(E->G != NULL);

        // Allocate a point in the group that will be used in setting the
        // generator. This is not needed after the generator is set.
        P = EC_POINT_new(E->G);
        GOTO_ERROR_UNLESS(P != NULL);

        // Need to use this in case Montgomery method is being used
        GOTO_ERROR_UNLESS(
            EC_POINT_set_affine_coordinates_GFp(E->G, P, bnX, bnY, CTX));
        // Now set the generator
        GOTO_ERROR_UNLESS(EC_GROUP_set_generator(E->G, P, bnN, bnH));

        EC_POINT_free(P);
        goto Exit;
Error:
        EC_POINT_free(P);
        BnCurveFree(E);
        E = NULL;
    }
Exit:
    return E;
}

//*** BnCurveFree()
// This function will free the allocated components of the curve and end the
// frame in which the curve data exists
LIB_EXPORT void BnCurveFree(bigCurveData* E)
{
    if(E)
    {
        EC_GROUP_free(E->G);
        OsslContextLeave(E->CTX);
    }
}

//*** BnEccModMult()
// This function does a point multiply of the form R = [d]S
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation; treat as result being point at infinity
LIB_EXPORT BOOL BnEccModMult(bigPoint   R,  // OUT: computed point
                             pointConst S,  // IN: point to multiply by 'd' (optional)
                             bigConst   d,  // IN: scalar for [d]S
                             const bigCurveData* E)
{
    EC_POINT* pR = EC_POINT_new(E->G);
    EC_POINT* pS = EcPointInitialized(S, E);
    BIG_INITIALIZED(bnD, d);

    if(S == NULL)
        EC_POINT_mul(E->G, pR, bnD, NULL, NULL, E->CTX);
    else
        EC_POINT_mul(E->G, pR, NULL, pS, bnD, E->CTX);
    PointFromOssl(R, pR, E);
    EC_POINT_free(pR);
    EC_POINT_free(pS);
    return !BnEqualZero(R->z);
}

//*** BnEccModMult2()
```

```
// This function does a point multiply of the form R = [d]G + [u]Q
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation; treat as result being point at infinity
LIB_EXPORT BOOL BnEccModMult2(bigPoint            R,  // OUT: computed point
                              pointConst          S,  // IN: optional point
                              bigConst            d,  // IN: scalar for [d]S or [d]G
                              pointConst          Q,  // IN: second point
                              bigConst            u,  // IN: second scalar
                              const bigCurveData* E   // IN: curve
)
{
    EC_POINT* pR = EC_POINT_new(E->G);
    EC_POINT* pS = EcPointInitialized(S, E);
    BIG_INITIALIZED(bnD, d);
    EC_POINT* pQ = EcPointInitialized(Q, E);
    BIG_INITIALIZED(bnU, u);

    if(S == NULL || S == (pointConst) & (AccessCurveConstants(E)->base))
        EC_POINT_mul(E->G, pR, bnD, pQ, bnU, E->CTX);
    else
    {
        const EC_POINT* points[2];
        const BIGNUM*   scalars[2];
        points[0]  = pS;
        points[1]  = pQ;
        scalars[0] = bnD;
        scalars[1] = bnU;
        EC_POINTs_mul(E->G, pR, NULL, 2, points, scalars, E->CTX);
    }
    PointFromOssl(R, pR, E);
    EC_POINT_free(pR);
    EC_POINT_free(pS);
    EC_POINT_free(pQ);
    return !BnEqualZero(R->z);
}

//** BnEccAdd()
// This function does addition of two points.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure in operation; treat as result being point at infinity
LIB_EXPORT BOOL BnEccAdd(bigPoint            R,  // OUT: computed point
                         pointConst          S,  // IN: first point to add
                         pointConst          Q,  // IN: second point
                         const bigCurveData* E   // IN: curve
)
{
    EC_POINT* pR = EC_POINT_new(E->G);
    EC_POINT* pS = EcPointInitialized(S, E);
    EC_POINT* pQ = EcPointInitialized(Q, E);
    //
    EC_POINT_add(E->G, pR, pS, pQ, E->CTX);

    PointFromOssl(R, pR, E);
    EC_POINT_free(pR);
    EC_POINT_free(pS);
    EC_POINT_free(pQ);
    return !BnEqualZero(R->z);
}

#  endif  // ALG_ECC

#  if CRYPTO_LIB_REPORTING

//** BnGetImplementation()
```

```
// This function reports the underlying library being used for bignum operations.
void BnGetImplementation(_CRYPTO_IMPL_DESCRIPTION* result)
{
    OsslGetVersion(result);
}

#  endif  // CRYPTO_LIB_REPORTING

#endif  // MATHLIB OSSL
```

/tpm/cryptolibs/Ossl/TpmToOsslSupport.c

```
//** Introduction
//
// The functions in this file are used for initialization of the interface to the
// OpenSSL library.

//** Defines and Includes

#include "BnOssl.h"
#include <CryptoInterface.h>
#include <Ossl/TpmToOsslSym.h>
#include <Ossl/TpmToOsslHash.h>
#include <openssl/opensslv.h>
#include <stdio.h>

#if CRYPTO_LIB_REPORTING

//*** OsslGetVersion()
// Report the version of OpenSSL.
void OsslGetVersion(_CRYPTO_IMPL_DESCRIPTION* result)
{
    snprintf(result->name, sizeof(result->name), "OpenSSL");
#  if defined(OPENSSL_VERSION_STR)
    snprintf(result->version, sizeof(result->version), "%s", OPENSSL_VERSION_STR);
#  else
    // decode the hex version string according to the rules described in opensslv.h
    snprintf(result->version,
             sizeof(result->version),
             "%d.%d.%d%c",
             (unsigned char)((OPENSSL_VERSION_NUMBER >> 28) & 0x0f),
             (unsigned char)((OPENSSL_VERSION_NUMBER >> 20) & 0xff),
             (unsigned char)((OPENSSL_VERSION_NUMBER >> 12) & 0xff),
             (char)((OPENSSL_VERSION_NUMBER >> 4) & 0xff) - 1 + 'a');
#  endif  //OPENSSL_VERSION_STR
}

#endif  //CRYPTO_LIB_REPORTING

#if defined(HASH_LIB_OSSL) || defined(MATH_LIB_OSSL) || defined(SYM_LIB_OSSL)
// Used to pass the pointers to the correct sub-keys
typedef const BYTE* desKeyPointers[3];

//*** BnSupportLibInit()
// This does any initialization required by the support library.
LIB_EXPORT int BnSupportLibInit(void)
{
    return TRUE;
}

//*** OsslContextEnter()
// This function is used to initialize an OpenSSL context at the start of a function
// that will call to an OpenSSL math function.
BN_CTX* OsslContextEnter(void)
{
```

```
    BN_CTX* CTX = BN_CTX_new();
    //
    return OsslPushContext(CTX);
}

//*** OsslContextLeave()
// This is the companion function to OsslContextEnter().
void OsslContextLeave(BN_CTX* CTX)
{
    OsslPopContext(CTX);
    BN_CTX_free(CTX);
}

//*** OsslPushContext()
// This function is used to create a frame in a context. All values allocated within
// this context after the frame is started will be automatically freed when the
// context (OsslPopContext()
BN_CTX* OsslPushContext(BN_CTX* CTX)
{
    if(CTX == NULL)
        FAIL(FATAL_ERROR_ALLOCATION);
    BN_CTX_start(CTX);
    return CTX;
}

//*** OsslPopContext()
// This is the companion function to OsslPushContext().
void OsslPopContext(BN_CTX* CTX)
{
    // BN_CTX_end can't be called with NULL. It will blow up.
    if(CTX != NULL)
        BN_CTX_end(CTX);
}

#  if CRYPTO_LIB_REPORTING

#    if defined(SYM_LIB_OSSL) && SIMULATION && CRYPTO_LIB_REPORTING
//*** _crypto_GetSymImpl()
// Report the version of OpenSSL being used for symmetric crypto.
void _crypto_GetSymImpl(_CRYPTO_IMPL_DESCRIPTION* result)
{
    OsslGetVersion(result);
}
#    else
#      error huh?
#    endif  // defined(SYM_LIB_OSSL) && SIMULATION

#    if defined(HASH_LIB_OSSL) && SIMULATION && CRYPTO_LIB_REPORTING
//*** _crypto_GetHashImpl()
// Report the version of OpenSSL being used for hashing.
void _crypto_GetHashImpl(_CRYPTO_IMPL_DESCRIPTION* result)
{
    OsslGetVersion(result);
}
#    endif  // defined(HASH_LIB_OSSL) && SIMULATION

#  endif  // CRYPTO_LIB_REPORTING

#endif  // HASH_LIB_OSSL || MATH_LIB_OSSL || SYM_LIB_OSSL
```

/tpm/cryptolibs/Ossl/include/BnOssl.h

```
//** Introduction
// This file contains the headers necessary to build the Open SSL support for
// the TpmBigNum library.
```

```
#ifndef _BNOSSL_H_
#define _BNOSSL_H_
// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/tpm_public.h>
#include <public/prototypes/TpmFail_fp.h>
#include <Ossl/BnToOsslMath.h>
// TODO_RENAME_INC_FOLDER: these refer to TpmBigNum protected headers
#include <BnSupport_Interface.h>
#include <BnUtil_fp.h>
#include <BnMemory_fp.h>
#include <BnMath_fp.h>
#include <BnConvert_fp.h>

#if CRYPTO_LIB_REPORTING
#  include <CryptoInterface.h>

//*** OsslGetVersion()
// Report the current version of OpenSSL.
void OsslGetVersion(_CRYPTO_IMPL_DESCRIPTION* result);

#endif  // CRYPTO_LIB_REPORTING

#endif  // _BNOSSL_H_
```

/tpm/cryptolibs/Ossl/include/Ossl/BnToOsslMath.h

```
//** Introduction
// This file contains OpenSSL specific functions called by TpmBigNum library to
provide
// the TpmBigNum + OpenSSL math support.

#ifndef _BN_TO_OSSL_MATH_H_
#define _BN_TO_OSSL_MATH_H_

#define MATH_LIB_OSSL

// Require TPM Big Num types
#if !defined(MATH_LIB_TPMBIGNUM) && !defined(_BNOSSL_H_)
#  error this OpenSSL Interface expects to be used from TpmBigNum
#endif

#include <BnValues.h>
#include <openssl/evp.h>
#include <openssl/ec.h>
#include <openssl/bn.h>

#if OPENSSL_VERSION_NUMBER >= 0x30100000L
// Check the bignum_st definition against the one below and either update the
// version check or provide the new definition for this version.
#  error Untested OpenSSL version
#elif OPENSSL_VERSION_NUMBER >= 0x10100000L
// from crypto/bn/bn_lcl.h (OpenSSL 1.x) or crypto/bn/bn_local.h (OpenSSL 3.0)
struct bignum_st
{
    BN_ULONG* d;   /* Pointer to an array of 'BN_BITS2' bit
                                * chunks. */
    int       top; /* Index of last used d +1. */
                   /* The next are internal book keeping for bn_expand. */
    int dmax;      /* Size of the d array. */
    int neg;       /* one if the number is negative */
    int flags;
};
#else
#  define EC_POINT_get_affine_coordinates EC_POINT_get_affine_coordinates_GFp
#  define EC_POINT_set_affine_coordinates EC_POINT_set_affine_coordinates_GFp
```

```
#endif  // OPENSSL_VERSION_NUMBER

//** Macros and Defines

// Make sure that the library is using the correct size for a crypt word
#if defined THIRTY_TWO_BIT && (RADIX_BITS != 32)                \
    || ((defined SIXTY_FOUR_BIT_LONG || defined SIXTY_FOUR_BIT) \
        && (RADIX_BITS != 64))
#  error Ossl library is using different radix
#endif

// Allocate a local BIGNUM value. For the allocation, a bigNum structure is created
// as is a local BIGNUM. The bigNum is initialized and then the BIGNUM is
// set to reference the local value.
#define BIG_VAR(name, bits)         \
    BN_VAR(name##Bn, (bits));       \
    BIGNUM  _##name;                \
    BIGNUM* name = BigInitialized( \
        &_##name, BnInit(name##Bn, BYTES_TO_CRYPT_WORDS(sizeof(_##name##Bn.d))))

// Allocate a BIGNUM and initialize with the values in a bigNum initializer
#define BIG_INITIALIZED(name, initializer) \
    BIGNUM  _##name;                       \
    BIGNUM* name = BigInitialized(&_##name, initializer)

typedef struct
{
    const TPMBN_ECC_CURVE_CONSTANTS* C;  // the TPM curve values
    EC_GROUP*                        G;  // group parameters
    BN_CTX* CTX;  // the context for the math (this might not be
                  // the context in which the curve was created>;
} OSSL_CURVE_DATA;

// Define the curve data type expected by the TpmBigNum library:
typedef OSSL_CURVE_DATA                      bigCurveData;

TPM_INLINE const TPMBN_ECC_CURVE_CONSTANTS* AccessCurveConstants(
    const bigCurveData* E)
{
    return E->C;
}

#include <Ossl/TpmToOsslSupport_fp.h>

// Start and end a context within which the OpenSSL memory management works
#define OSSL_ENTER() BN_CTX* CTX = OsslContextEnter()
#define OSSL_LEAVE() OsslContextLeave(CTX)

// Start and end a local stack frame within the context of the curve frame
#define ECC_ENTER() BN_CTX* CTX = OsslPushContext(E->CTX)
#define ECC_LEAVE() OsslPopContext(CTX)

#define BN_NEW() BnNewVariable(CTX)

// This definition would change if there were something to report
#define MathLibSimulationEnd()

#endif  // _BN_TO_OSSL_MATH_H_
```

/tpm/cryptolibs/Ossl/include/Ossl/BnToOsslMath_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Oct 24, 2019  Time: 11:37:07AM
 */
```

```
#ifndef _BN_TO_OSSL_MATH_FP_H_
#define _BN_TO_OSSL_MATH_FP_H_

#ifdef MATH_LIB_OSSL

//*** OsslToTpmBn()
// This function converts an OpenSSL BIGNUM to a TPM bigNum. In this implementation
// it is assumed that OpenSSL uses a different control structure but the same data
// layout -- an array of native-endian words in little-endian order.
//  Return Type: BOOL
//      TRUE(1)         success
//      FALSE(0)        failure because value will not fit or OpenSSL variable doesn't
//                      exist
BOOL OsslToTpmBn(bigNum bn, BIGNUM* osslBn);

//*** BigInitialized()
// This function initializes an OSSL BIGNUM from a TPM bigConst. Do not use this for
// values that are passed to OpenSLL when they are not declared as const in the
// function prototype. Instead, use BnNewVariable().
BIGNUM* BigInitialized(BIGNUM* toInit, bigConst initializer);
#endif  // MATHLIB OSSL

#endif  // _TPM_TO_OSSL_MATH_FP_H_
```

/tpm/cryptolibs/Ossl/include/Ossl/TpmToOsslHash.h

```
//** Introduction
//
// This header file is used to 'splice' the OpenSSL hash code into the TPM code.
//
#ifndef HASH_LIB_DEFINED
#define HASH_LIB_DEFINED

#define HASH_LIB_OSSL

#include <openssl/evp.h>
#include <openssl/sha.h>

#if ALG_SM3_256
#  if defined(OPENSSL_NO_SM3) || OPENSSL_VERSION_NUMBER < 0x10101010L
#    error "Current version of OpenSSL doesn't support SM3"
#  elif OPENSSL_VERSION_NUMBER >= 0x10200000L
#    include <openssl/sm3.h>
#  else
// OpenSSL 1.1.1 keeps smX.h headers in the include/crypto directory,
// and they do not get installed as part of the libssl package
#    define SM3_LBLOCK (64 / 4)

typedef struct SM3state_st
{
    unsigned int A, B, C, D, E, F, G, H;
    unsigned int Nl, Nh;
    unsigned int data[SM3_LBLOCK];
    unsigned int num;
} SM3_CTX;

int sm3_init(SM3_CTX* c);
int sm3_update(SM3_CTX* c, const void* data, size_t len);
int sm3_final(unsigned char* md, SM3_CTX* c);
#  endif  // OpenSSL < 1.2
#endif    // ALG_SM3_256

#include <openssl/ossl_typ.h>
```

```
//*****************************************************************
//** Links to the OpenSSL HASH code
//*****************************************************************

// Redefine the internal name used for each of the hash state structures to the
// name used by the library.
// These defines need to be known in all parts of the TPM so that the structure
// sizes can be properly computed when needed.
#define tpmHashStateSHA1_t    SHA_CTX
#define tpmHashStateSHA256_t  SHA256_CTX
#define tpmHashStateSHA384_t  SHA512_CTX
#define tpmHashStateSHA512_t  SHA512_CTX
#define tpmHashStateSM3_256_t SM3_CTX

// The defines below are only needed when compiling CryptHash.c or CryptSmac.c.
// This isolation is primarily to avoid name space collision. However, if there
// is a real collision, it will likely show up when the linker tries to put things
// together.

#ifdef _CRYPT_HASH_C_

typedef BYTE*       PBYTE;
typedef const BYTE* PCBYTE;

// Define the interface between CryptHash.c to the functions provided by the
// library. For each method, define the calling parameters of the method and then
// define how the method is invoked in CryptHash.c.
//
// All hashes are required to have the same calling sequence. If they don't, create
// a simple adaptation function that converts from the "standard" form of the call
// to the form used by the specific hash (and then send a nasty letter to the
// person who wrote the hash function for the library).
//
// The macro that calls the method also defines how the
// parameters get swizzled between the default form (in CryptHash.c)and the
// library form.
//
// Initialize the hash context
#  define HASH_START_METHOD_DEF void(HASH_START_METHOD)(PANY_HASH_STATE state)
#  define HASH_START(hashState) ((hashState)->def->method.start)(&(hashState)->state);

// Add data to the hash
#  define HASH_DATA_METHOD_DEF \
      void(HASH_DATA_METHOD)(PANY_HASH_STATE state, PCBYTE buffer, size_t size)
#  define HASH_DATA(hashState, dInSize, dIn) \
      ((hashState)->def->method.data)(&(hashState)->state, dIn, dInSize)

// Finalize the hash and get the digest
#  define HASH_END_METHOD_DEF \
      void(HASH_END_METHOD)(BYTE * buffer, PANY_HASH_STATE state)
#  define HASH_END(hashState, buffer) \
      ((hashState)->def->method.end)(buffer, &(hashState)->state)

// Copy the hash context
// Note: For import, export, and copy, memcpy() is used since there is no
// reformatting necessary between the internal and external forms.
#  define HASH_STATE_COPY_METHOD_DEF \
      void(HASH_STATE_COPY_METHOD)(  \
          PANY_HASH_STATE to, PCANY_HASH_STATE from, size_t size)
#  define HASH_STATE_COPY(hashStateOut, hashStateIn)            \
      ((hashStateIn)->def->method.copy)(&(hashStateOut)->state, \
                                        &(hashStateIn)->state,  \
                                        (hashStateIn)->def->contextSize)

// Copy (with reformatting when necessary) an internal hash structure to an
// external blob
```

```
#  define HASH_STATE_EXPORT_METHOD_DEF \
      void(HASH_STATE_EXPORT_METHOD)(BYTE * to, PCANY_HASH_STATE from, size_t size)
#  define HASH_STATE_EXPORT(to, hashStateFrom)            \
      ((hashStateFrom)->def->method.copyOut)(             \
          &(((BYTE*)(to))[offsetof(HASH_STATE, state)]), \
          &(hashStateFrom)->state,                        \
          (hashStateFrom)->def->contextSize)

// Copy from an external blob to an internal formate (with reformatting when
// necessary
#  define HASH_STATE_IMPORT_METHOD_DEF \
      void(HASH_STATE_IMPORT_METHOD)(  \
          PANY_HASH_STATE to, const BYTE* from, size_t size)
#  define HASH_STATE_IMPORT(hashStateTo, from)                    \
      ((hashStateTo)->def->method.copyIn)(                        \
          &(hashStateTo)->state,                                  \
          &(((const BYTE*)(from))[offsetof(HASH_STATE, state)]), \
          (hashStateTo)->def->contextSize)

// Function aliases. The code in CryptHash.c uses the internal designation for the
// functions. These need to be translated to the function names of the library.
#  define tpmHashStart_SHA1          SHA1_Init
#  define tpmHashData_SHA1           SHA1_Update
#  define tpmHashEnd_SHA1            SHA1_Final
#  define tpmHashStateCopy_SHA1      memcpy
#  define tpmHashStateExport_SHA1    memcpy
#  define tpmHashStateImport_SHA1    memcpy
#  define tpmHashStart_SHA256        SHA256_Init
#  define tpmHashData_SHA256         SHA256_Update
#  define tpmHashEnd_SHA256          SHA256_Final
#  define tpmHashStateCopy_SHA256    memcpy
#  define tpmHashStateExport_SHA256  memcpy
#  define tpmHashStateImport_SHA256  memcpy
#  define tpmHashStart_SHA384        SHA384_Init
#  define tpmHashData_SHA384         SHA384_Update
#  define tpmHashEnd_SHA384          SHA384_Final
#  define tpmHashStateCopy_SHA384    memcpy
#  define tpmHashStateExport_SHA384  memcpy
#  define tpmHashStateImport_SHA384  memcpy
#  define tpmHashStart_SHA512        SHA512_Init
#  define tpmHashData_SHA512         SHA512_Update
#  define tpmHashEnd_SHA512          SHA512_Final
#  define tpmHashStateCopy_SHA512    memcpy
#  define tpmHashStateExport_SHA512  memcpy
#  define tpmHashStateImport_SHA512  memcpy
#  define tpmHashStart_SM3_256       sm3_init
#  define tpmHashData_SM3_256        sm3_update
#  define tpmHashEnd_SM3_256         sm3_final
#  define tpmHashStateCopy_SM3_256   memcpy
#  define tpmHashStateExport_SM3_256 memcpy
#  define tpmHashStateImport_SM3_256 memcpy

#endif  // _CRYPT_HASH_C_

#define LibHashInit()
// This definition would change if there were something to report
#define HashLibSimulationEnd()

#endif  // HASH_LIB_DEFINED
```

/tpm/cryptolibs/Ossl/include/Ossl/TpmToOsslSupport_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar 28, 2019  Time: 08:25:19PM
```

```
 */

#ifndef _TPM_TO_OSSL_SUPPORT_FP_H_
#define _TPM_TO_OSSL_SUPPORT_FP_H_

#if defined(HASH_LIB_OSSL) || defined(MATH_LIB_OSSL) || defined(SYM_LIB_OSSL)

//*** BnSupportLibInit()
// This does any initialization required by the support library.
LIB_EXPORT int BnSupportLibInit(void);

//*** OsslContextEnter()
// This function is used to initialize an OpenSSL context at the start of a function
// that will call to an OpenSSL math function.
BN_CTX* OsslContextEnter(void);

//*** OsslContextLeave()
// This is the companion function to OsslContextEnter().
void OsslContextLeave(BN_CTX* CTX);

//*** OsslPushContext()
// This function is used to create a frame in a context. All values allocated within
// this context after the frame is started will be automatically freed when the
// context (OsslPopContext()
BN_CTX* OsslPushContext(BN_CTX* CTX);

//*** OsslPopContext()
// This is the companion function to OsslPushContext().
void OsslPopContext(BN_CTX* CTX);
#endif  // HASH_LIB_OSSL || MATH_LIB_OSSL || SYM_LIB_OSSL

#endif  // _TPM_TO_OSSL_SUPPORT_FP_H_
```

/tpm/cryptolibs/Ossl/include/Ossl/TpmToOsslSym.h

```
//** Introduction
//
// This header file is used to 'splice' the OpenSSL library into the TPM code.
//
// The support required of a library are a hash module, a block cipher module and
// portions of a big number library.

// All of the library-dependent headers should have the same guard to that only the
// first one gets defined.
#ifndef SYM_LIB_DEFINED
#define SYM_LIB_DEFINED

#define SYM_LIB_OSSL

#include <openssl/aes.h>

#if ALG_SM4
#  if defined(OPENSSL_NO_SM4) || OPENSSL_VERSION_NUMBER < 0x10101010L
#    error "Current version of OpenSSL doesn't support SM4"
#  elif OPENSSL_VERSION_NUMBER >= 0x10200000L
#    include <openssl/sm4.h>
#  else
// OpenSSL 1.1.1 keeps smX.h headers in the include/crypto directory,
// and they do not get installed as part of the libssl package

#    define SM4_KEY_SCHEDULE 32

typedef struct SM4_KEY_st
{
    uint32_t rk[SM4_KEY_SCHEDULE];
```

```
} SM4_KEY;

int  SM4_set_key(const uint8_t* key, SM4_KEY* ks);
void SM4_encrypt(const uint8_t* in, uint8_t* out, const SM4_KEY* ks);
void SM4_decrypt(const uint8_t* in, uint8_t* out, const SM4_KEY* ks);
#  endif  // OpenSSL < 1.2
#endif    // ALG_SM4

#if ALG_CAMELLIA
#  include <openssl/camellia.h>
#endif

#include <openssl/bn.h>
#include <openssl/ossl_typ.h>

//**********************************************************************
//** Links to the OpenSSL symmetric algorithms.
//**********************************************************************

// The Crypt functions that call the block encryption function use the parameters
// in the order:
//  1) keySchedule
//  2) in buffer
//  3) out buffer
// Since open SSL uses the order in encryptoCall_t above, need to swizzle the
// values to the order required by the library.
#define SWIZZLE(keySchedule, in, out) \
    (const BYTE*)(in), (BYTE*)(out), (void*)(keySchedule)

// Define the order of parameters to the library functions that do block encryption
// and decryption.
typedef void (*TpmCryptSetSymKeyCall_t)(const BYTE* in, BYTE* out, void* keySchedule);

//**********************************************************************
//** Links to the OpenSSL AES code
//**********************************************************************
// Macros to set up the encryption/decryption key schedules
//
// AES:
#define TpmCryptSetEncryptKeyAES(key, keySizeInBits, schedule) \
    AES_set_encrypt_key((key), (keySizeInBits), (tpmKeyScheduleAES*)(schedule))
#define TpmCryptSetDecryptKeyAES(key, keySizeInBits, schedule) \
    AES_set_decrypt_key((key), (keySizeInBits), (tpmKeyScheduleAES*)(schedule))

// Macros to alias encryption calls to specific algorithms. This should be used
// sparingly. Currently, only used by CryptSym.c and CryptRand.c
//
// When using these calls, to call the AES block encryption code, the caller
// should use:
//      TpmCryptEncryptAES(SWIZZLE(keySchedule, in, out));
#define TpmCryptEncryptAES AES_encrypt
#define TpmCryptDecryptAES AES_decrypt
#define tpmKeyScheduleAES  AES_KEY

//**********************************************************************
//** Links to the OpenSSL SM4 code
//**********************************************************************
// Macros to set up the encryption/decryption key schedules
#define TpmCryptSetEncryptKeySM4(key, keySizeInBits, schedule) \
    SM4_set_key((key), (tpmKeyScheduleSM4*)(schedule))
#define TpmCryptSetDecryptKeySM4(key, keySizeInBits, schedule) \
    SM4_set_key((key), (tpmKeyScheduleSM4*)(schedule))

// Macros to alias encryption calls to specific algorithms. This should be used
// sparingly.
#define TpmCryptEncryptSM4 SM4_encrypt
```

```
#define TpmCryptDecryptSM4 SM4_decrypt
#define tpmKeyScheduleSM4  SM4_KEY

//**********************************************************************
//** Links to the OpenSSL CAMELLIA code
//**********************************************************************
// Macros to set up the encryption/decryption key schedules
#define TpmCryptSetEncryptKeyCAMELLIA(key, keySizeInBits, schedule) \
    Camellia_set_key((key), (keySizeInBits), (tpmKeyScheduleCAMELLIA*)(schedule))
#define TpmCryptSetDecryptKeyCAMELLIA(key, keySizeInBits, schedule) \
    Camellia_set_key((key), (keySizeInBits), (tpmKeyScheduleCAMELLIA*)(schedule))

// Macros to alias encryption calls to specific algorithms. This should be used
// sparingly.
#define TpmCryptEncryptCAMELLIA Camellia_encrypt
#define TpmCryptDecryptCAMELLIA Camellia_decrypt
#define tpmKeyScheduleCAMELLIA  CAMELLIA_KEY

// Forward reference

typedef union tpmCryptKeySchedule_t tpmCryptKeySchedule_t;

// This definition would change if there were something to report
#define SymLibSimulationEnd()

#endif  // SYM_LIB_DEFINED
```

Annex C
(informative)
Simulation Environment

C.1 Introduction

These files are used to simulate some of the implementation-dependent hardware of a TPM. These files are provided to allow creation of a simulation environment for the TPM. These files are not expected to be part of a hardware TPM implementation.

/Platform/include/Platform.h

```
#ifndef _PLATFORM_H_
#define _PLATFORM_H_

#include <TpmConfiguration/TpmBuildSwitches.h>
#include <TpmConfiguration/TpmProfile.h>
// TODO_RENAME_INC_FOLDER: public refers to the TPM_CoreLib public headers
#include <public/BaseTypes.h>
#include <public/TPMB.h>
#include <public/MinMax.h>

#include "PlatformACT.h"
#include "PlatformClock.h"
#include "PlatformData.h"
#include "prototypes/platform_public_interface.h"
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/tpm_to_platform_interface.h>
#include <platform_interface/platform_to_tpm_interface.h>

#define GLOBAL_C
#define NV_C
#include <platform_interface/pcrstruct.h>
#include <platform_interface/prototypes/platform_pcr_fp.h>

#endif  // _PLATFORM_H_
```

/Platform/include/PlatformACT.h

```
// This file contains the definitions for the ACT macros and data types used in the
// ACT implementation.

#ifndef _PLATFORM_ACT_H_
#define _PLATFORM_ACT_H_

typedef struct ACT_DATA
{
    uint32_t remaining;
    uint32_t newValue;
    uint8_t  signaled;
    uint8_t  pending;
    uint8_t  number;
} ACT_DATA, *P_ACT_DATA;

#if !(defined RH_ACT_0) || (RH_ACT_0 != YES)
#  undef RH_ACT_0
#  define RH_ACT_0 NO
#  define IF_ACT_0_IMPLEMENTED(op)
#else
#  define IF_ACT_0_IMPLEMENTED(op) op(0)
```

```
#endif
#if !(defined RH_ACT_1) || (RH_ACT_1 != YES)
#  undef RH_ACT_1
#  define RH_ACT_1 NO
#  define IF_ACT_1_IMPLEMENTED(op)
#else
#  define IF_ACT_1_IMPLEMENTED(op) op(1)
#endif
#if !(defined RH_ACT_2) || (RH_ACT_2 != YES)
#  undef RH_ACT_2
#  define RH_ACT_2 NO
#  define IF_ACT_2_IMPLEMENTED(op)
#else
#  define IF_ACT_2_IMPLEMENTED(op) op(2)
#endif
#if !(defined RH_ACT_3) || (RH_ACT_3 != YES)
#  undef RH_ACT_3
#  define RH_ACT_3 NO
#  define IF_ACT_3_IMPLEMENTED(op)
#else
#  define IF_ACT_3_IMPLEMENTED(op) op(3)
#endif
#if !(defined RH_ACT_4) || (RH_ACT_4 != YES)
#  undef RH_ACT_4
#  define RH_ACT_4 NO
#  define IF_ACT_4_IMPLEMENTED(op)
#else
#  define IF_ACT_4_IMPLEMENTED(op) op(4)
#endif
#if !(defined RH_ACT_5) || (RH_ACT_5 != YES)
#  undef RH_ACT_5
#  define RH_ACT_5 NO
#  define IF_ACT_5_IMPLEMENTED(op)
#else
#  define IF_ACT_5_IMPLEMENTED(op) op(5)
#endif
#if !(defined RH_ACT_6) || (RH_ACT_6 != YES)
#  undef RH_ACT_6
#  define RH_ACT_6 NO
#  define IF_ACT_6_IMPLEMENTED(op)
#else
#  define IF_ACT_6_IMPLEMENTED(op) op(6)
#endif
#if !(defined RH_ACT_7) || (RH_ACT_7 != YES)
#  undef RH_ACT_7
#  define RH_ACT_7 NO
#  define IF_ACT_7_IMPLEMENTED(op)
#else
#  define IF_ACT_7_IMPLEMENTED(op) op(7)
#endif
#if !(defined RH_ACT_8) || (RH_ACT_8 != YES)
#  undef RH_ACT_8
#  define RH_ACT_8 NO
#  define IF_ACT_8_IMPLEMENTED(op)
#else
#  define IF_ACT_8_IMPLEMENTED(op) op(8)
#endif
#if !(defined RH_ACT_9) || (RH_ACT_9 != YES)
#  undef RH_ACT_9
#  define RH_ACT_9 NO
#  define IF_ACT_9_IMPLEMENTED(op)
#else
#  define IF_ACT_9_IMPLEMENTED(op) op(9)
#endif
#if !(defined RH_ACT_A) || (RH_ACT_A != YES)
#  undef RH_ACT_A
```

```
#  define RH_ACT_A NO
#  define IF_ACT_A_IMPLEMENTED(op)
#else
#  define IF_ACT_A_IMPLEMENTED(op) op(A)
#endif
#if !(defined RH_ACT_B) || (RH_ACT_B != YES)
#  undef RH_ACT_B
#  define RH_ACT_B NO
#  define IF_ACT_B_IMPLEMENTED(op)
#else
#  define IF_ACT_B_IMPLEMENTED(op) op(B)
#endif
#if !(defined RH_ACT_C) || (RH_ACT_C != YES)
#  undef RH_ACT_C
#  define RH_ACT_C NO
#  define IF_ACT_C_IMPLEMENTED(op)
#else
#  define IF_ACT_C_IMPLEMENTED(op) op(C)
#endif
#if !(defined RH_ACT_D) || (RH_ACT_D != YES)
#  undef RH_ACT_D
#  define RH_ACT_D NO
#  define IF_ACT_D_IMPLEMENTED(op)
#else
#  define IF_ACT_D_IMPLEMENTED(op) op(D)
#endif
#if !(defined RH_ACT_E) || (RH_ACT_E != YES)
#  undef RH_ACT_E
#  define RH_ACT_E NO
#  define IF_ACT_E_IMPLEMENTED(op)
#else
#  define IF_ACT_E_IMPLEMENTED(op) op(E)
#endif
#if !(defined RH_ACT_F) || (RH_ACT_F != YES)
#  undef RH_ACT_F
#  define RH_ACT_F NO
#  define IF_ACT_F_IMPLEMENTED(op)
#else
#  define IF_ACT_F_IMPLEMENTED(op) op(F)
#endif

#define FOR_EACH_ACT(op)      \
    IF_ACT_0_IMPLEMENTED(op) \
    IF_ACT_1_IMPLEMENTED(op) \
    IF_ACT_2_IMPLEMENTED(op) \
    IF_ACT_3_IMPLEMENTED(op) \
    IF_ACT_4_IMPLEMENTED(op) \
    IF_ACT_5_IMPLEMENTED(op) \
    IF_ACT_6_IMPLEMENTED(op) \
    IF_ACT_7_IMPLEMENTED(op) \
    IF_ACT_8_IMPLEMENTED(op) \
    IF_ACT_9_IMPLEMENTED(op) \
    IF_ACT_A_IMPLEMENTED(op) \
    IF_ACT_B_IMPLEMENTED(op) \
    IF_ACT_C_IMPLEMENTED(op) \
    IF_ACT_D_IMPLEMENTED(op) \
    IF_ACT_E_IMPLEMENTED(op) \
    IF_ACT_F_IMPLEMENTED(op)

#endif  // _PLATFORM_ACT_H_
```

/Platform/include/PlatformClock.h

```
// This file contains the instance data for the Platform module. It is collected
// in this file so that the state of the module is easier to manage.
```

```
#ifndef _PLATFORM_CLOCK_H_
#define _PLATFORM_CLOCK_H_

#ifndef _ARM_
#  ifdef _MSC_VER
#    include <sys/types.h>
#    include <sys/timeb.h>
#  else
#    include <time.h>
#  endif
#endif

#endif  // _PLATFORM_CLOCK_H_
```

/Platform/include/PlatformData.h

```
// This file contains the instance data for the Platform module. It is collected
// in this file so that the state of the module is easier to manage.

#ifndef _PLATFORM_DATA_H_
#define _PLATFORM_DATA_H_

#ifndef EXTERN
#  ifdef _PLATFORM_DATA_C_
#    define EXTERN
#  else
#    define EXTERN extern
#  endif  // _PLATFORM_DATA_C_
#endif    // EXTERN

// From Cancel.c
// Cancel flag.  It is initialized as FALSE, which indicate the command is not
// being canceled
EXTERN int s_isCanceled;

#ifndef HARDWARE_CLOCK
typedef uint64_t clock64_t;
// This is the value returned the last time that the system clock was read. This
// is only relevant for a simulator or virtual TPM.
EXTERN clock64_t s_realTimePrevious;

// These values are used to try to synthesize a long lived version of clock().
EXTERN clock64_t s_lastSystemTime;
EXTERN clock64_t s_lastReportedTime;

// This is the rate adjusted value that is the equivalent of what would be read from
// a hardware register that produced rate adjusted time.
EXTERN clock64_t s_tpmTime;
#endif  // HARDWARE_CLOCK

// This value indicates that the timer was reset
EXTERN int s_timerReset;
// This value indicates that the timer was stopped. It causes a clock discontinuity.
EXTERN int s_timerStopped;

// This variable records the time when _plat__TimerReset is called.  This mechanism
// allow us to subtract the time when TPM is power off from the total
// time reported by clock() function
EXTERN uint64_t s_initClock;

// This variable records the timer adjustment factor.
EXTERN unsigned int s_adjustRate;

// For LocalityPlat.c
```

```
// Locality of current command
EXTERN unsigned char s_locality;

// For NVMem.c
// Choose if the NV memory should be backed by RAM or by file.
// If this macro is defined, then a file is used as NV.  If it is not defined,
// then RAM is used to back NV memory. Comment out to use RAM.

#if(!defined VTPM) || ((VTPM != NO) && (VTPM != YES))
#  undef VTPM
#  define VTPM YES  // Default: Either YES or NO
#endif

// For a simulation, use a file to back up the NV
#if(!defined FILE_BACKED_NV) || ((FILE_BACKED_NV != NO) && (FILE_BACKED_NV != YES))
#  undef FILE_BACKED_NV
#  define FILE_BACKED_NV (VTPM && YES)  // Default: Either YES or NO
#endif

#if SIMULATION
#  undef FILE_BACKED_NV
#  define FILE_BACKED_NV YES
#endif  // SIMULATION

EXTERN unsigned char s_NV[NV_MEMORY_SIZE];
EXTERN int           s_NvIsAvailable;
EXTERN int           s_NV_unrecoverable;
EXTERN int           s_NV_recoverable;

// For PPPlat.c
// Physical presence.  It is initialized to FALSE
EXTERN int s_physicalPresence;

// From Power
EXTERN int s_powerLost;

// For Entropy.c
EXTERN uint32_t lastEntropy;

#define DEFINE_ACT(N) EXTERN ACT_DATA ACT_##N;
FOR_EACH_ACT(DEFINE_ACT)

EXTERN int actTicksAllowed;

#endif  // _PLATFORM_DATA_H_
```

/Platform/include/prototypes/platform_public_interface.h

```
// This file contains the interface into the platform layer from external callers.
// External callers are expected to be implementation specific, and may be a simulator
// or some other implementation

#ifndef _PLATFORM_PUBLIC_INTERFACE_H_
#define _PLATFORM_PUBLIC_INTERFACE_H_

#include <stddef.h>

//** From Cancel.c

// Set cancel flag.
LIB_EXPORT void _plat__SetCancel(void);

//***_plat__ClearCancel()
// Clear cancel flag
LIB_EXPORT void _plat__ClearCancel(void);
```

```
//** From Clock.c

//***_plat__TimerReset()
// This function sets current system clock time as t0 for counting TPM time.
// This function is called at a power on event to reset the clock. When the clock
// is reset, the indication that the clock was stopped is also set.
LIB_EXPORT void _plat__TimerReset(void);

//*** _plat__TimerRestart()
// This function should be called in order to simulate the restart of the timer
// should it be stopped while power is still applied.
LIB_EXPORT void _plat__TimerRestart(void);

//*** _plat__RealTime()
// This is another, probably futile, attempt to define a portable function
// that will return a 64-bit clock value that has mSec resolution.
LIB_EXPORT uint64_t _plat__RealTime(void);

//** From LocalityPlat.c

//***_plat__LocalitySet()
// Set the most recent command locality in locality value form
LIB_EXPORT void _plat__LocalitySet(unsigned char locality);

//** From NVMem.c

//*** _plat__NvErrors()
// This function is used by the simulator to set the error flags in the NV
// subsystem to simulate an error in the NV loading process
LIB_EXPORT void _plat__NvErrors(int recoverable, int unrecoverable);

//***_plat__NVDisable()
// Disable NV memory
LIB_EXPORT void _plat__NVDisable(
    void*  platParameter,  // platform specific parameter
    size_t paramSize       // size of parameter. If size == 0, then
                           // parameter is a sizeof(void*) scalar and should
                           // be cast to an integer (intptr_t), not dereferenced.
);

//***_plat__SetNvAvail()
// Set the current NV state to available.  This function is for testing purpose
// only.  It is not part of the platform NV logic
LIB_EXPORT void _plat__SetNvAvail(void);

//***_plat__ClearNvAvail()
// Set the current NV state to unavailable.  This function is for testing purpose
// only.  It is not part of the platform NV logic
LIB_EXPORT void _plat__ClearNvAvail(void);

//*** _plat__NVNeedsManufacture()
// This function is used by the simulator to determine when the TPM's NV state
// needs to be manufactured.
LIB_EXPORT int _plat__NVNeedsManufacture(void);

//** From PlatformACT.c

//*** _plat__ACT_GetPending()
LIB_EXPORT int _plat__ACT_GetPending(uint32_t act  //IN: number of ACT to check
);

//*** _plat__ACT_Tick()
// This processes the once-per-second clock tick from the hardware. This is set up
// for the simulator to use the control interface to send ticks to the TPM. These
// ticks do not have to be on a per second basis. They can be as slow or as fast as
```

```
// desired so that the simulation can be tested.
LIB_EXPORT void _plat__ACT_Tick(void);

//** From PowerPlat.c

//*** _plat__Signal_PowerOn()
// Signal platform power on
LIB_EXPORT int _plat__Signal_PowerOn(void);

//*** _plat_Signal_Reset()
// This a TPM reset without a power loss.
LIB_EXPORT int _plat__Signal_Reset(void);

//*** _plat__Signal_PowerOff()
// Signal platform power off
LIB_EXPORT void _plat__Signal_PowerOff(void);

//** From PPPlat.c

//*** _plat__Signal_PhysicalPresenceOn()
// Signal physical presence on
LIB_EXPORT void _plat__Signal_PhysicalPresenceOn(void);

//*** _plat__Signal_PhysicalPresenceOff()
// Signal physical presence off
LIB_EXPORT void _plat__Signal_PhysicalPresenceOff(void);

//*** _plat__SetTpmFirmwareHash()
// Called by the simulator to set the TPM Firmware hash used for
// firmware-bound hierarchies. Not a cryptographically-strong hash.
#if SIMULATION
LIB_EXPORT void _plat__SetTpmFirmwareHash(uint32_t hash);
#endif

//*** _plat__SetTpmFirmwareSvn()
// Called by the simulator to set the TPM Firmware SVN reported by
// getCapability.
#if SIMULATION
LIB_EXPORT void _plat__SetTpmFirmwareSvn(uint16_t svn);
#endif

//** From RunCommand.c

//*** _plat__RunCommand()
// This version of RunCommand will set up a jum_buf and call ExecuteCommand(). If
// the command executes without failing, it will return and RunCommand will return.
// If there is a failure in the command, then _plat__Fail() is called and it will
// longjump back to RunCommand which will call ExecuteCommand again. However, this
// time, the TPM will be in failure mode so ExecuteCommand will simply build
// a failure response and return.
LIB_EXPORT void _plat__RunCommand(
    uint32_t        requestSize,   // IN: command buffer size
    unsigned char*  request,       // IN: command buffer
    uint32_t*       responseSize,  // IN/OUT: response buffer size
    unsigned char** response       // IN/OUT: response buffer
);

#endif  // _PLATFORM_PUBLIC_INTERFACE_H_
```

/Platform/src/Cancel.c

```
//** Description
//
//  This module simulates the cancel pins on the TPM.
//
```

```
//** Includes, Typedefs, Structures, and Defines
#include "Platform.h"

//** Functions

//***_plat__IsCanceled()
// Check if the cancel flag is set
//  Return Type: int
//      TRUE(1)         if cancel flag is set
//      FALSE(0)        if cancel flag is not set
LIB_EXPORT int _plat__IsCanceled(void)
{
    // return cancel flag
    return s_isCanceled;
}

//***_plat__SetCancel()

// Set cancel flag.
LIB_EXPORT void _plat__SetCancel(void)
{
    s_isCanceled = TRUE;
    return;
}

//***_plat__ClearCancel()
// Clear cancel flag
LIB_EXPORT void _plat__ClearCancel(void)
{
    s_isCanceled = FALSE;
    return;
}
```

/Platform/src/Clock.c

```
//** Description
//
// This file contains the routines that are used by the simulator to mimic
// a hardware clock on a TPM.
//
// In this implementation, all the time values are measured in millisecond.
// However, the precision of the clock functions may be implementation dependent.

//** Includes and Data Definitions
#include <assert.h>
#include "Platform.h"

// CLOCK_NOMINAL is the number of hardware ticks per ms. A value of 30000 means
// that the nominal clock rate used to drive the hardware clock is 30 MHz. The
// adjustment rates are used to determine the conversion of the hardware ticks to
// internal hardware clock value. In practice, we would expect that there would be
// a hardware register will accumulated mS. It would be incremented by the output
// of a pre-scaler. The pre-scaler would divide the ticks from the clock by some
// value that would compensate for the difference between clock time and real time.
// The code in Clock does the emulation of this function.
#define CLOCK_NOMINAL 30000
// A 1% change in rate is 300 counts
#define CLOCK_ADJUST_COARSE 300
// A 0.1% change in rate is 30 counts
#define CLOCK_ADJUST_MEDIUM 30
// A minimum change in rate is 1 count
#define CLOCK_ADJUST_FINE 1
// The clock tolerance is +/-15% (4500 counts)
// Allow some guard band (16.7%)
#define CLOCK_ADJUST_LIMIT 5000
```

```
//** Simulator Functions
//*** Introduction
// This set of functions is intended to be called by the simulator environment in
// order to simulate hardware events.

//***_plat__TimerReset()
// This function sets current system clock time as t0 for counting TPM time.
// This function is called at a power on event to reset the clock. When the clock
// is reset, the indication that the clock was stopped is also set.
LIB_EXPORT void _plat__TimerReset(void)
{
    s_lastSystemTime = 0;
    s_tpmTime        = 0;
    s_adjustRate     = CLOCK_NOMINAL;
    s_timerReset     = TRUE;
    s_timerStopped   = TRUE;
    return;
}

//*** _plat__TimerRestart()
// This function should be called in order to simulate the restart of the timer
// should it be stopped while power is still applied.
LIB_EXPORT void _plat__TimerRestart(void)
{
    s_timerStopped = TRUE;
    return;
}

//** Functions Used by TPM
//*** Introduction
// These functions are called by the TPM code. They should be replaced by
// appropriated hardware functions.

#include <time.h>
clock_t debugTime;

//*** _plat__RealTime()
// This is another, probably futile, attempt to define a portable function
// that will return a 64-bit clock value that has mSec resolution.
LIB_EXPORT uint64_t _plat__RealTime(void)
{
    clock64_t time;
#ifdef _MSC_VER
    struct _timeb sysTime;
    //
    _ftime_s(&sysTime);
    time = (clock64_t)(sysTime.time) * 1000 + sysTime.millitm;
    // set the time back by one hour if daylight savings
    if(sysTime.dstflag)
        time -= 1000 * 60 * 60;  // mSec/sec * sec/min * min/hour = ms/hour
#else
    // hopefully, this will work with most UNIX systems
    struct timespec systime;
    //
    clock_gettime(CLOCK_MONOTONIC, &systime);
    time = (clock64_t)systime.tv_sec * 1000 + (systime.tv_nsec / 1000000);
#endif
    return time;
}

//***_plat__TimerRead()
// This function provides access to the tick timer of the platform. The TPM code
// uses this value to drive the TPM Clock.
//
// The tick timer is supposed to run when power is applied to the device. This timer
```

```
// should not be reset by time events including _TPM_Init. It should only be reset
// when TPM power is re-applied.
//
// If the TPM is run in a protected environment, that environment may provide the
// tick time to the TPM as long as the time provided by the environment is not
// allowed to go backwards. If the time provided by the system can go backwards
// during a power discontinuity, then the _plat__Signal_PowerOn should call
// _plat__TimerReset().
LIB_EXPORT uint64_t _plat__TimerRead(void)
{
#ifdef HARDWARE_CLOCK
#  error "need a defintion for reading the hardware clock"
    return HARDWARE_CLOCK
#else
    clock64_t timeDiff;
    clock64_t adjustedTimeDiff;
    clock64_t timeNow;
    clock64_t readjustedTimeDiff;

    // This produces a timeNow that is basically locked to the system clock.
    timeNow = _plat__RealTime();

    // if this hasn't been initialized, initialize it
    if(s_lastSystemTime == 0)
    {
        s_lastSystemTime   = timeNow;
        debugTime          = clock();
        s_lastReportedTime = 0;
        s_realTimePrevious = 0;
    }
    // The system time can bounce around and that's OK as long as we don't allow
    // time to go backwards. When the time does appear to go backwards, set
    // lastSystemTime to be the new value and then update the reported time.
    if(timeNow < s_lastReportedTime)
        s_lastSystemTime = timeNow;
    s_lastReportedTime = s_lastReportedTime + timeNow - s_lastSystemTime;
    s_lastSystemTime   = timeNow;
    timeNow            = s_lastReportedTime;

    // The code above produces a timeNow that is similar to the value returned
    // by Clock(). The difference is that timeNow does not max out, and it is
    // at a ms. rate rather than at a CLOCKS_PER_SEC rate. The code below
    // uses that value and does the rate adjustment on the time value.
    // If there is no difference in time, then skip all the computations
    if(s_realTimePrevious >= timeNow)
        return s_tpmTime;
    // Compute the amount of time since the last update of the system clock
    timeDiff = timeNow - s_realTimePrevious;

    // Do the time rate adjustment and conversion from CLOCKS_PER_SEC to mSec
    adjustedTimeDiff = (timeDiff * CLOCK_NOMINAL) / ((uint64_t)s_adjustRate);

    // update the TPM time with the adjusted timeDiff
    s_tpmTime += (clock64_t)adjustedTimeDiff;

    // Might have some rounding error that would loose CLOCKS. See what is not
    // being used. As mentioned above, this could result in putting back more than
    // is taken out. Here, we are trying to recreate timeDiff.
    readjustedTimeDiff = (adjustedTimeDiff * (uint64_t)s_adjustRate) / CLOCK_NOMINAL;

    // adjusted is now converted back to being the amount we should advance the
    // previous sampled time. It should always be less than or equal to timeDiff.
    // That is, we could not have use more time than we started with.
    s_realTimePrevious = s_realTimePrevious + readjustedTimeDiff;

#  ifdef DEBUGGING_TIME
```

```
    // Put this in so that TPM time will pass much faster than real time when
    // doing debug.
    // A value of 1000 for DEBUG_TIME_MULTIPLER will make each ms into a second
    // A good value might be 100
    return (s_tpmTime * DEBUG_TIME_MULTIPLIER);
#  endif
    return s_tpmTime;
#endif
}

//*** _plat__TimerWasReset()
// This function is used to interrogate the flag indicating if the tick timer has
// been reset.
//
// If the resetFlag parameter is SET, then the flag will be CLEAR before the
// function returns.
LIB_EXPORT int _plat__TimerWasReset(void)
{
    int retVal   = s_timerReset;
    s_timerReset = FALSE;
    return retVal;
}

//*** _plat__TimerWasStopped()
// This function is used to interrogate the flag indicating if the tick timer has
// been stopped. If so, this is typically a reason to roll the nonce.
//
// This function will CLEAR the s_timerStopped flag before returning. This provides
// functionality that is similar to status register that is cleared when read. This
// is the model used here because it is the one that has the most impact on the TPM
// code as the flag can only be accessed by one entity in the TPM. Any other
// implementation of the hardware can be made to look like a read-once register.
LIB_EXPORT int _plat__TimerWasStopped(void)
{
    int retVal     = s_timerStopped;
    s_timerStopped = FALSE;
    return retVal;
}

//***_plat__ClockAdjustRate()
// Adjust the clock rate
LIB_EXPORT void _plat__ClockRateAdjust(_plat__ClockAdjustStep adjust)
{
    // We expect the caller should only use a fixed set of constant values to
    // adjust the rate
    switch(adjust)
    {
        // slower increases the divisor
        case PLAT_TPM_CLOCK_ADJUST_COARSE_SLOWER:
            s_adjustRate += CLOCK_ADJUST_COARSE;
            break;
        case PLAT_TPM_CLOCK_ADJUST_MEDIUM_SLOWER:
            s_adjustRate += CLOCK_ADJUST_MEDIUM;
            break;
        case PLAT_TPM_CLOCK_ADJUST_FINE_SLOWER:
            s_adjustRate += CLOCK_ADJUST_FINE;
            break;
        // faster decreases the divisor
        case PLAT_TPM_CLOCK_ADJUST_FINE_FASTER:
            s_adjustRate -= CLOCK_ADJUST_FINE;
            break;
        case PLAT_TPM_CLOCK_ADJUST_MEDIUM_FASTER:
            s_adjustRate -= CLOCK_ADJUST_MEDIUM;
            break;
        case PLAT_TPM_CLOCK_ADJUST_COARSE_FASTER:
            s_adjustRate -= CLOCK_ADJUST_COARSE;
```

```
            break;
    }

    if(s_adjustRate > (CLOCK_NOMINAL + CLOCK_ADJUST_LIMIT))
        s_adjustRate = CLOCK_NOMINAL + CLOCK_ADJUST_LIMIT;
    if(s_adjustRate < (CLOCK_NOMINAL - CLOCK_ADJUST_LIMIT))
        s_adjustRate = CLOCK_NOMINAL - CLOCK_ADJUST_LIMIT;

    return;
}
```

/Platform/src/DebugHelpers.c

```
//** Description
//
//    This file contains the NV read and write access methods.  This implementation
//    uses RAM/file and does not manage the RAM/file as NV blocks.
//    The implementation may become more sophisticated over time.
//

//** Includes and Local
#include <stdio.h>
#include <time.h>
#include "Platform.h"

#if CERTIFYX509_DEBUG

const char* debugFileName = "DebugFile.txt";

//*** fileOpen()
// This exists to allow use of the 'safe' version of fopen() with a MS runtime.
static FILE* fileOpen(const char* fn, const char* mode)
{
    FILE*     f;
#  if defined _MSC_VER
    if(fopen_s(&f, fn, mode) != 0)
        f = NULL;
#  else
    f = fopen(fn, mode);
#  endif
    return f;
}

//*** DebugFileInit()
// This function initializes the file containing the debug data with the time of the
// file creation.
//  Return Type: int
//   0              success
//  != 0            error
int DebugFileInit(void)
{
    FILE*  f = NULL;
    time_t t = time(NULL);
//
// Get current date and time.
#  if defined _MSC_VER
    char      timeString[100];
    ctime_s(timeString, (size_t)sizeof(timeString), &t);
#  else
    char* timeString;
    timeString = ctime(&t);
#  endif
    // Try to open the debug file
    f = fileOpen(debugFileName, "w");
    if(f)
```

```
    {
        // Initialize the contents with the time.
        fprintf(f, "%s\n", timeString);
        fclose(f);
        return 0;
    }
    return -1;
}

//*** DebugDumpBuffer()
void DebugDumpBuffer(int size, unsigned char* buf, const char* identifier)
{
    int i;
    //
    FILE* f = fileOpen(debugFileName, "a");
    if(!f)
        return;
    if(identifier)
        fprintf(f, "%s\n", identifier);
    if(buf)
    {
        for(i = 0; i < size; i++)
        {
            if(((i % 16) == 0) && (i))
                fprintf(f, "\n");
            fprintf(f, " %02X", buf[i]);
        }
        if((size % 16) != 0)
            fprintf(f, "\n");
    }
    fclose(f);
}

#endif  // CERTIFYX509_DEBUG
```

/Platform/src/Entropy.c

```
//** Includes and Local Values

#define _CRT_RAND_S
#include <stdlib.h>
#include <memory.h>
#include <time.h>
#include "Platform.h"

#ifdef _MSC_VER
#  include <process.h>
#else
#  include <unistd.h>
#endif

// This is the last 32-bits of hardware entropy produced. We have to check to
// see that two consecutive 32-bit values are not the same because
// according to FIPS 140-2, annex C:
//
// "If each call to an RNG produces blocks of n bits (where n > 15), the first
// n-bit block generated after power-up, initialization, or reset shall not be
// used, but shall be saved for comparison with the next n-bit block to be
// generated. Each subsequent generation of an n-bit block shall be compared with
// the previously generated block. The test shall fail if any two compared n-bit
// blocks are equal."
extern uint32_t lastEntropy;

//** Functions
```

```
//*** rand32()
// Local function to get a 32-bit random number
static uint32_t rand32(void)
{
    uint32_t rndNum = rand();
#if RAND_MAX < UINT16_MAX
    // If the maximum value of the random number is a 15-bit number, then shift it up
    // 15 bits, get 15 more bits, shift that up 2 and then XOR in another value to get
    // a full 32 bits.
    rndNum = (rndNum << 15) ^ rand();
    rndNum = (rndNum << 2) ^ rand();
#elif RAND_MAX == UINT16_MAX
    // If the maximum size is 16-bits, shift it and add another 16 bits
    rndNum = (rndNum << 16) ^ rand();
#elif RAND_MAX < UINT32_MAX
    // If 31 bits, then shift 1 and include another random value to get the extra bit
    rndNum = (rndNum << 1) ^ rand();
#endif
    return rndNum;
}

//*** _plat__GetEntropy()
// This function is used to get available hardware entropy. In a hardware
// implementation of this function, there would be no call to the system
// to get entropy.
//  Return Type: int32_t
//  < 0         hardware failure of the entropy generator, this is sticky
// >= 0         the returned amount of entropy (bytes)
//
LIB_EXPORT int32_t _plat__GetEntropy(unsigned char* entropy,  // output buffer
                                     uint32_t       amount    // amount requested
)
{
    uint32_t rndNum;
    int32_t  ret;
    //
    if(amount == 0)
    {
        // Seed the platform entropy source if the entropy source is software. There
        // is no reason to put a guard macro (#if or #ifdef) around this code because
        // this code would not be here if someone was changing it for a system with
        // actual hardware.
        //
        // NOTE 1: The following command does not provide proper cryptographic
        // entropy. Its primary purpose to make sure that different instances of the
        // simulator, possibly started by a script on the same machine, are seeded
        // differently. Vendors of the actual TPMs need to ensure availability of
        // proper entropy using their platform-specific means.
        //
        // NOTE 2: In debug builds by default the reference implementation will seed
        // its RNG deterministically (without using any platform provided randomness).
        // See the USE_DEBUG_RNG macro and DRBG_GetEntropy() function.
#ifdef _MSC_VER
        srand((unsigned)_plat__RealTime() ^ _getpid());
#else
        srand((unsigned)_plat__RealTime() ^ getpid());
#endif
        lastEntropy = rand32();
        ret         = 0;
    }
    else
    {
        rndNum = rand32();
        if(rndNum == lastEntropy)
        {
            ret = -1;
```

```
        }
        else
        {
            lastEntropy = rndNum;
            // Each process will have its random number generator initialized
            // according to the process id and the initialization time. This is not a
            // lot of entropy so, to add a bit more, XOR the current time value into
            // the returned entropy value.
            // NOTE: the reason for including the time here rather than have it in
            // in the value assigned to lastEntropy is that rand() could be broken and
            // using the time would in the lastEntropy value would hide this.
            rndNum ^= (uint32_t)_plat__RealTime();

            // Only provide entropy 32 bits at a time to test the ability
            // of the caller to deal with partial results.
            ret = MIN(amount, sizeof(rndNum));
            memcpy(entropy, &rndNum, ret);
        }
    }
    return ret;
}
```

/Platform/src/ExtraData.c

```
//** Description
//
// This file contains routines that are called by the core library to allow the
// platform to use the Core storage structures for small amounts of related data.
//
// In this implementation, the buffers are all just set to 0xFF

//** Includes and Data Definitions
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include "Platform.h"

//** _plat__GetPlatformManufactureData

// This function allows the platform to provide a small amount of data to be
// stored as part of the TPM's PERSISTENT_DATA structure during manufacture.  Of
// course the platform can store data separately as well, but this allows a
// simple platform implementation to store a few bytes of data without
// implementing a multi-layer storage system.  This function is called on
// manufacture and CLEAR.  The buffer will contain the last value provided
// to the Core library.
LIB_EXPORT void _plat__GetPlatformManufactureData(uint8_t* pPlatformPersistentData,
                                                  uint32_t bufferSize)
{
    if(bufferSize != 0)
    {
        memset((void*)pPlatformPersistentData, 0xFF, bufferSize);
    }
}
```

/Platform/src/LocalityPlat.c

```
//** Includes
#include "Platform.h"

//** Functions

//***_plat__LocalityGet()
// Get the most recent command locality in locality value form.
```

```
// This is an integer value for locality and not a locality structure
// The locality can be 0-4 or 32-255. 5-31 is not allowed.
LIB_EXPORT unsigned char _plat__LocalityGet(void)
{
    return s_locality;
}

//***_plat__LocalitySet()
// Set the most recent command locality in locality value form
LIB_EXPORT void _plat__LocalitySet(unsigned char locality)
{
    if(locality > 4 && locality < 32)
        locality = 0;
    s_locality = locality;
    return;
}
```

/Platform/src/NVMem.c

```
//** Description
//
//    This file contains the NV read and write access methods.  This implementation
//    uses RAM/file and does not manage the RAM/file as NV blocks.
//    The implementation may become more sophisticated over time.
//

//** Includes and Local
#include <memory.h>
#include <string.h>
#include <assert.h>
#include "Platform.h"
#if FILE_BACKED_NV
#  include <stdio.h>
static FILE* s_NvFile           = NULL;
static int   s_NeedsManufacture = FALSE;
#endif

//**Functions

#if FILE_BACKED_NV
const char* s_NvFilePath = "NVChip";

//*** NvFileOpen()
// This function opens the file used to hold the NV image.
//  Return Type: int
//  >= 0        success
//  -1          error
static int NvFileOpen(const char* mode)
{
    // Try to open an exist NVChip file for read/write
#  if defined _MSC_VER && 1
    if(fopen_s(&s_NvFile, s_NvFilePath, mode) != 0)
    {
        s_NvFile = NULL;
    }
#  else
    s_NvFile = fopen(s_NvFilePath, mode);
#  endif
    return (s_NvFile == NULL) ? -1 : 0;
}

//*** NvFileCommit()
// Write all of the contents of the NV image to a file.
//  Return Type: int
//      TRUE(1)         success
```

```
//      FALSE(0)        failure
static int NvFileCommit(void)
{
    int OK;
    // If NV file is not available, return failure
    if(s_NvFile == NULL)
        return 1;
    // Write RAM data to NV
    fseek(s_NvFile, 0, SEEK_SET);
    OK = (NV_MEMORY_SIZE == fwrite(s_NV, 1, NV_MEMORY_SIZE, s_NvFile));
    OK = OK && (0 == fflush(s_NvFile));
    assert(OK);
    return OK;
}

//*** NvFileSize()
// This function gets the size of the NV file and puts the file pointer where desired
// using the seek method values. SEEK_SET => beginning; SEEK_CUR => current position
// and SEEK_END => to the end of the file.
static long NvFileSize(int leaveAt)
{
    long fileSize;
    long filePos = ftell(s_NvFile);
    //
    assert(NULL != s_NvFile);

    int fseek_result = fseek(s_NvFile, 0, SEEK_END);
    NOT_REFERENCED(fseek_result);  // Fix compiler warning for NDEBUG
    assert(fseek_result == 0);
    fileSize = ftell(s_NvFile);
    assert(fileSize >= 0);
    switch(leaveAt)
    {
        case SEEK_SET:
            filePos = 0;
        case SEEK_CUR:
            fseek(s_NvFile, filePos, SEEK_SET);
            break;
        case SEEK_END:
            break;
        default:
            assert(FALSE);
            break;
    }
    return fileSize;
}
#endif

//*** _plat__NvErrors()
// This function is used by the simulator to set the error flags in the NV
// subsystem to simulate an error in the NV loading process
LIB_EXPORT void _plat__NvErrors(int recoverable, int unrecoverable)
{
    s_NV_unrecoverable = unrecoverable;
    s_NV_recoverable   = recoverable;
}

//***_plat__NVEnable()
// Enable NV memory.
//
// This version just pulls in data from a file. In a real TPM, with NV on chip,
// this function would verify the integrity of the saved context. If the NV
// memory was not on chip but was in something like RPMB, the NV state would be
// read in, decrypted and integrity checked.
//
// The recovery from an integrity failure depends on where the error occurred. It
```

```
// it was in the state that is discarded by TPM Reset, then the error is
// recoverable if the TPM is reset. Otherwise, the TPM must go into failure mode.
//  Return Type: int
//      0           if success
//      > 0         if receive recoverable error
//      <0          if unrecoverable error
#define NV_ENABLE_SUCCESS 0
#define NV_ENABLE_FAILED  (-1)
LIB_EXPORT int _plat__NVEnable(
    void*  platParameter,  // platform specific parameter
    size_t paramSize       // size of parameter. If size == 0, then
                           // parameter is a sizeof(void*) scalar and should
                           // be cast to an integer (intptr_t), not dereferenced.
)
{
    NOT_REFERENCED(platParameter);  // to keep compiler quiet
    NOT_REFERENCED(paramSize);      // to keep compiler quiet

    // Start assuming everything is OK
    s_NV_unrecoverable = FALSE;
    s_NV_recoverable   = FALSE;
#if FILE_BACKED_NV
    if(s_NvFile != NULL)
        return NV_ENABLE_SUCCESS;
    // Initialize all the bytes in the ram copy of the NV
    _plat__NvMemoryClear(0, NV_MEMORY_SIZE);

    // If the file exists
    if(NvFileOpen("r+b") >= 0)
    {
        long fileSize = NvFileSize(SEEK_SET);  // get the file size and leave the
                                               // file pointer at the start
                                               //
        // If the size is right, read the data
        if(NV_MEMORY_SIZE == fileSize)
        {
            s_NeedsManufacture = fread(s_NV, 1, NV_MEMORY_SIZE, s_NvFile)
                                 != NV_MEMORY_SIZE;
        }
        else
        {
            NvFileCommit();  // for any other size, initialize it
            s_NeedsManufacture = TRUE;
        }
    }
    // If NVChip file does not exist, try to create it for read/write.
    else if(NvFileOpen("w+b") >= 0)
    {
        NvFileCommit();  // Initialize the file
        s_NeedsManufacture = TRUE;
    }
    assert(NULL != s_NvFile);  // Just in case we are broken for some reason.
#endif
    // NV contents have been initialized and the error checks have been performed. For
    // simulation purposes, use the signaling interface to indicate if an error is
    // to be simulated and the type of the error.
    if(s_NV_unrecoverable)
        return NV_ENABLE_FAILED;
    s_NvIsAvailable = TRUE;
    return s_NV_recoverable;
}

//***_plat__NVDisable()
// Disable NV memory
LIB_EXPORT void _plat__NVDisable(
    void*  platParameter,  // platform specific parameter
```

```
    size_t paramSize       // size of parameter. If size == 0, then
                           // parameter is a sizeof(void*) scalar and should
                           // be cast to an integer (intptr_t), not dereferenced.
)
{
    NOT_REFERENCED(paramSize);  // to keep compiler quiet
    int delete = ((intptr_t)platParameter != 0)
                    ? TRUE
                    : FALSE;  // IN: If TRUE (!=0), delete the NV contents.

#if FILE_BACKED_NV
    if(NULL != s_NvFile)
    {
        fclose(s_NvFile);  // Close NV file
        // Alternative to deleting the file is to set its size to 0. This will not
        // match the NV size so the TPM will need to be remanufactured.
        if(delete)
        {
            // Open for writing at the start. Sets the size to zero.
            if(NvFileOpen("w") >= 0)
            {
                fflush(s_NvFile);
                fclose(s_NvFile);
            }
        }
    }
    s_NvFile = NULL;  // Set file handle to NULL
#endif
    s_NvIsAvailable = FALSE;
    return;
}

//***_plat__GetNvReadyState()
// Check if NV is available
//  Return Type: int
//      0               NV is available
//      1               NV is not available due to write failure
//      2               NV is not available due to rate limit
LIB_EXPORT int _plat__GetNvReadyState(void)
{
    int retVal = NV_READY;
    if(!s_NvIsAvailable)
        retVal = NV_WRITEFAILURE;
#if FILE_BACKED_NV
    else
        retVal = (s_NvFile == NULL);
#endif
    return retVal;
}

//***_plat__NvMemoryRead()
// Function: Read a chunk of NV memory
//  Return Type: int
//      TRUE(1)         offset and size is within available NV size
//      FALSE(0)        otherwise; also trigger failure mode
LIB_EXPORT int _plat__NvMemoryRead(unsigned int startOffset,  // IN: read start
                                   unsigned int size,  // IN: size of bytes to read
                                   void*        data   // OUT: data buffer
)
{
    assert(startOffset + size <= NV_MEMORY_SIZE);
    if(startOffset + size <= NV_MEMORY_SIZE)
    {
        memcpy(data, &s_NV[startOffset], size);  // Copy data from RAM
        return TRUE;
    }
```

```
    return FALSE;
}

//*** _plat__NvGetChangedStatus()
// This function checks to see if the NV is different from the test value. This is
// so that NV will not be written if it has not changed.
//  Return Type: int
//      NV_HAS_CHANGED(1)       the NV location is different from the test value
//      NV_IS_SAME(0)           the NV location is the same as the test value
//      NV_INVALID_LOCATION(-1) the NV location is invalid; also triggers failure mode
LIB_EXPORT int _plat__NvGetChangedStatus(
    unsigned int startOffset,  // IN: read start
    unsigned int size,         // IN: size of bytes to read
    void*        data          // IN: data buffer
)
{
    assert(startOffset + size <= NV_MEMORY_SIZE);
    if(startOffset + size <= NV_MEMORY_SIZE)
    {
        return (memcmp(&s_NV[startOffset], data, size) != 0);
    }
    // the NV location is invalid; the assert above should have triggered failure
    // mode
    return NV_INVALID_LOCATION;
}

//***_plat__NvMemoryWrite()
// This function is used to update NV memory. The "write" is to a memory copy of
// NV. At the end of the current command, any changes are written to
// the actual NV memory.
// NOTE: A useful optimization would be for this code to compare the current
// contents of NV with the local copy and note the blocks that have changed. Then
// only write those blocks when _plat__NvCommit() is called.
//  Return Type: int
//      TRUE(1)         offset and size is within available NV size
//      FALSE(0)        otherwise; also trigger failure mode
LIB_EXPORT int _plat__NvMemoryWrite(unsigned int startOffset,  // IN: write start
                                    unsigned int size,  // IN: size of bytes to write
                                    void*        data   // OUT: data buffer
)
{
    assert(startOffset + size <= NV_MEMORY_SIZE);
    if(startOffset + size <= NV_MEMORY_SIZE)
    {
        memcpy(&s_NV[startOffset], data, size);  // Copy the data to the NV image
        return TRUE;
    }
    return FALSE;
}

//***_plat__NvMemoryClear()
// Function is used to set a range of NV memory bytes to an implementation-dependent
// value. The value represents the erase state of the memory.
LIB_EXPORT int _plat__NvMemoryClear(unsigned int startOffset,  // IN: clear start
                                    unsigned int size  // IN: number of bytes to clear
)
{
    assert(startOffset + size <= NV_MEMORY_SIZE);
    if(startOffset + size <= NV_MEMORY_SIZE)
    {
        // In this implementation, assume that the erase value for NV is all 1s
        memset(&s_NV[startOffset], 0xff, size);
        return TRUE;
    }
    return FALSE;
}
```

```
//***_plat__NvMemoryMove()
// Function: Move a chunk of NV memory from source to destination
//      This function should ensure that if there overlap, the original data is
//      copied before it is written
LIB_EXPORT int _plat__NvMemoryMove(unsigned int sourceOffset,  // IN: source offset
                                   unsigned int destOffset,  // IN: destination offset
                                   unsigned int size  // IN: size of data being moved
)
{
    assert(sourceOffset + size <= NV_MEMORY_SIZE);
    assert(destOffset + size <= NV_MEMORY_SIZE);
    if(sourceOffset + size <= NV_MEMORY_SIZE && destOffset + size <= NV_MEMORY_SIZE)
    {
        memmove(&s_NV[destOffset], &s_NV[sourceOffset], size);  // Move data in RAM
        return TRUE;
    }
    return FALSE;
}

//***_plat__NvCommit()
// This function writes the local copy of NV to NV for permanent store. It will write
// NV_MEMORY_SIZE bytes to NV. If a file is use, the entire file is written.
//  Return Type: int
//  0       NV write success
//  non-0   NV write fail
LIB_EXPORT int _plat__NvCommit(void)
{
#if FILE_BACKED_NV
    return (NvFileCommit() ? 0 : 1);
#else
    return 0;
#endif
}

//***_plat__TearDown
// notify platform that TPM_TearDown was called so platform can cleanup or
// zeroize anything in the Platform.  This should zeroize NV as well.
LIB_EXPORT void _plat__TearDown()
{
#if FILE_BACKED_NV
    // remove(s_NvFilePath);
#endif
}

//***_plat__SetNvAvail()
// Set the current NV state to available.  This function is for testing purpose
// only.  It is not part of the platform NV logic
LIB_EXPORT void _plat__SetNvAvail(void)
{
    s_NvIsAvailable = TRUE;
    return;
}

//***_plat__ClearNvAvail()
// Set the current NV state to unavailable.  This function is for testing purpose
// only.  It is not part of the platform NV logic
LIB_EXPORT void _plat__ClearNvAvail(void)
{
    s_NvIsAvailable = FALSE;
    return;
}

//*** _plat__NVNeedsManufacture()
// This function is used by the simulator to determine when the TPM's NV state
// needs to be manufactured.
```

```
LIB_EXPORT int _plat__NVNeedsManufacture(void)
{
#if FILE_BACKED_NV
    return s_NeedsManufacture;
#else
    return FALSE;
#endif
}
```

/Platform/src/PlatformACT.c

```
//** Includes
#include "Platform.h"

//** Functions

#if ACT_SUPPORT

//*** ActSignal()
// Function called when there is an ACT event to signal or unsignal
static void ActSignal(P_ACT_DATA actData, int on)
{
    if(actData == NULL)
        return;
    // If this is to turn a signal on, don't do anything if it is already on. If this
    // is to turn the signal off, do it anyway because this might be for
    // initialization.
    if(on && (actData->signaled == TRUE))
        return;
    actData->signaled = (uint8_t)on;

    // If there is an action, then replace the "Do something" with the correct action.
    // It should test 'on' to see if it is turning the signal on or off.
    switch(actData->number)
    {
#   if RH_ACT_0
        case 0:  // Do something
            return;
#   endif
#   if RH_ACT_1
        case 1:  // Do something
            return;
#   endif
#   if RH_ACT_2
        case 2:  // Do something
            return;
#   endif
#   if RH_ACT_3
        case 3:  // Do something
            return;
#   endif
#   if RH_ACT_4
        case 4:  // Do something
            return;
#   endif
#   if RH_ACT_5
        case 5:  // Do something
            return;
#   endif
#   if RH_ACT_6
        case 6:  // Do something
            return;
#   endif
#   if RH_ACT_7
        case 7:  // Do something
```

```
            return;
#   endif
#   if RH_ACT_8
        case 8:  // Do something
            return;
#   endif
#   if RH_ACT_9
        case 9:  // Do something
            return;
#   endif
#   if RH_ACT_A
        case 0xA:  // Do something
            return;
#   endif
#   if RH_ACT_B
        case 0xB:
            // Do something
            return;
#   endif
#   if RH_ACT_C
        case 0xC:  // Do something
            return;
#   endif
#   if RH_ACT_D
        case 0xD:  // Do something
            return;
#   endif
#   if RH_ACT_E
        case 0xE:  // Do something
            return;
#   endif
#   if RH_ACT_F
        case 0xF:  // Do something
            return;
#   endif
        default:
            return;
    }
}

//*** ActGetDataPointer()
static P_ACT_DATA ActGetDataPointer(uint32_t act)
{

#   define RETURN_ACT_POINTER(N) \
      if(0x##N == act)           \
          return &ACT_##N;

    FOR_EACH_ACT(RETURN_ACT_POINTER)

    return (P_ACT_DATA)NULL;
}

//*** _plat__ACT_GetImplemented()
// This function tests to see if an ACT is implemented. It is a belt and suspenders
// function because the TPM should not be calling to manipulate an ACT that is not
// implemented. However, this could help the simulator code which doesn't necessarily
// know if an ACT is implemented or not.
LIB_EXPORT int _plat__ACT_GetImplemented(uint32_t act)
{
    return (ActGetDataPointer(act) != NULL);
}

//*** _plat__ACT_GetRemaining()
// This function returns the remaining time. If an update is pending, 'newValue' is
// returned. Otherwise, the current counter value is returned. Note that since the
```

```
// timers keep running, the returned value can get stale immediately. The actual count
// value will be no greater than the returned value.
LIB_EXPORT uint32_t _plat__ACT_GetRemaining(uint32_t act  //IN: the ACT selector
)
{
    P_ACT_DATA actData = ActGetDataPointer(act);
    uint32_t   remain;
    //
    if(actData == NULL)
        return 0;
    remain = actData->remaining;
    if(actData->pending)
        remain = actData->newValue;
    return remain;
}

//*** _plat__ACT_GetSignaled()
LIB_EXPORT int _plat__ACT_GetSignaled(uint32_t act  //IN: number of ACT to check
)
{
    P_ACT_DATA actData = ActGetDataPointer(act);
    //
    if(actData == NULL)
        return 0;
    return (int)actData->signaled;
}

//*** _plat__ACT_SetSignaled()
LIB_EXPORT void _plat__ACT_SetSignaled(uint32_t act, int on)
{
    ActSignal(ActGetDataPointer(act), on);
}

//*** _plat__ACT_GetPending()
LIB_EXPORT int _plat__ACT_GetPending(uint32_t act  //IN: number of ACT to check
)
{
    P_ACT_DATA actData = ActGetDataPointer(act);
    //
    if(actData == NULL)
        return 0;
    return (int)actData->pending;
}

//*** _plat__ACT_UpdateCounter()
// This function is used to write the newValue for the counter. If an update is
// pending, then no update occurs and the function returns FALSE. If 'setSignaled'
// is TRUE, then the ACT signaled state is SET and if 'newValue' is 0, nothing
// is posted.
LIB_EXPORT int _plat__ACT_UpdateCounter(uint32_t act,      // IN: ACT to update
                                        uint32_t newValue  // IN: the value to post
)
{
    P_ACT_DATA actData = ActGetDataPointer(act);
    //
    if(actData == NULL)
        // actData doesn't exist but pretend update is pending rather than indicate
        // that a retry is necessary.
        return TRUE;
    // if an update is pending then return FALSE so that there will be a retry
    if(actData->pending != 0)
        return FALSE;
    actData->newValue = newValue;
    actData->pending  = TRUE;

    return TRUE;
```

```
}

//***_plat__ACT_EnableTicks()
// This enables and disables the processing of the once-per-second ticks. This should
// be turned off ('enable' = FALSE) by _TPM_Init and turned on ('enable' = TRUE) by
// TPM2_Startup() after all the initializations have completed.
LIB_EXPORT void _plat__ACT_EnableTicks(int enable)
{
    actTicksAllowed = enable;
}

//*** ActDecrement()
// If 'newValue' is non-zero it is copied to 'remaining' and then 'newValue' is
// set to zero. Then 'remaining' is decremented by one if it is not already zero. If
// the value is decremented to zero, then the associated event is signaled. If setting
// 'remaining' causes it to be greater than 1, then the signal associated with the ACT
// is turned off.
static void ActDecrement(P_ACT_DATA actData)
{
    // Check to see if there is an update pending
    if(actData->pending)
    {
        // If this update will cause the count to go from non-zero to zero, set
        // the newValue to 1 so that it will timeout when decremented below.
        if((actData->newValue == 0) && (actData->remaining != 0))
            actData->newValue = 1;
        actData->remaining = actData->newValue;

        // Update processed
        actData->pending = 0;
    }
    // no update so countdown if the count is non-zero but not max
    if((actData->remaining != 0) && (actData->remaining != UINT32_MAX))
    {
        // If this countdown causes the count to go to zero, then turn the signal for
        // the ACT on.
        if((actData->remaining -= 1) == 0)
            ActSignal(actData, TRUE);
    }
    // If the current value of the counter is non-zero, then the signal should be
    // off.
    if(actData->signaled && (actData->remaining > 0))
        ActSignal(actData, FALSE);
}

//*** _plat__ACT_Tick()
// This processes the once-per-second clock tick from the hardware. This is set up
// for the simulator to use the control interface to send ticks to the TPM. These
// ticks do not have to be on a per second basis. They can be as slow or as fast as
// desired so that the simulation can be tested.
LIB_EXPORT void _plat__ACT_Tick(void)
{
    // Ticks processing is turned off at certain times just to make sure that nothing
    // strange is happening before pointers and things are
    if(actTicksAllowed)
    {
        // Handle the update for each counter.
#   define DECREMENT_COUNT(N) ActDecrement(&ACT_##N);

        FOR_EACH_ACT(DECREMENT_COUNT)
    }
}

//*** ActZero()
// This function initializes a single ACT
static void ActZero(uint32_t act, P_ACT_DATA actData)
```

```
{
    actData->remaining = 0;
    actData->newValue  = 0;
    actData->pending   = 0;
    actData->number    = (uint8_t)act;
    ActSignal(actData, FALSE);
}

//***_plat__ACT_Initialize()
// This function initializes the ACT hardware and data structures
LIB_EXPORT int _plat__ACT_Initialize(void)
{
    actTicksAllowed = 0;
#  define ZERO_ACT(N) ActZero(0x##N, &ACT_##N);
    FOR_EACH_ACT(ZERO_ACT)

    return TRUE;
}

#endif  // ACT_SUPPORT
```

/Platform/src/PlatformData.c

```
//** Description
// This file will instance the TPM variables that are not stack allocated. The
// descriptions for these variables are in Global.h for this project.

//** Includes
#define _PLATFORM_DATA_C_
#include "Platform.h"
```

/Platform/src/PlatformPcr.c

```
// PCR platform interface functions
#include "Platform.h"
#include <public/TpmAlgorithmDefines.h>

// use this as a convenient lookup for hash size for PCRs.
UINT16 CryptHashGetDigestSize(TPM_ALG_ID hashAlg  // IN: hash algorithm to look up
);
void   MemorySet(void* dest, int value, size_t size);

// The initial value of PCR attributes.  The value of these fields should be
// consistent with PC Client specification.  The bitfield meanings are defined by
// the TPM Reference code.
// In this implementation, we assume the total number of implemented PCR is 24.
static const PCR_Attributes s_initAttributes[] = {
    //
    // PCR 0 - 15, static RTM
    // PCR[0]
    {
        1,  // save state
        0,  // in the "do not increment the PcrCounter" group? (0 = increment the
PcrCounter)
        0,  // supportsPolicyAuth group number? 0 = policyAuth not supported for this
PCR.
        0,  // supportsAuthValue group number? 0 = AuthValue not supported for this
PCR.
        0,    // 0 = reset localities (cannot reset)
        0x1F  // 0x1F = extendlocalities [0,4]
    },
    {1, 0, 0, 0, 0, 0x1F},  // PCR 1-3
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},
```

```
    {1, 0, 0, 0, 0, 0x1F},  // PCR 4-6
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},  // PCR 7-9
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},  // PCR 10-12
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},  // PCR 13-15
    {1, 0, 0, 0, 0, 0x1F},
    {1, 0, 0, 0, 0, 0x1F},

    // these PCRs are never saved
    {0, 0, 0, 0, 0x0F, 0x1F},  // PCR 16, Debug, reset allowed, extend all
    {0, 0, 0, 0, 0x10, 0x1C},  // PCR 17, Locality 4, extend loc 2+
    {0, 0, 0, 0, 0x10, 0x1C},  // PCR 18, Locality 3, extend loc 2+
    {0, 0, 0, 0, 0x10, 0x0C},  // PCR 19, Locality 2, extend loc 2, 3
    // these three support doNotIncrement, PolicyAuth, and AuthValue.
    // this is consistent with the existing behavior of the TPM Reference code
    // but differs from the behavior of the PC client spec.
    {0, 1, 1, 1, 0x14, 0x0E},  // PCR 20, Locality 1, extend loc 1, 2, 3
    {0, 1, 1, 1, 0x14, 0x04},  // PCR 21, Dynamic OS, extend loc 2
    {0, 1, 1, 1, 0x14, 0x04},  // PCR 22, Dynamic OS, extend loc 2
    {0, 0, 0, 0, 0x0F, 0x1F},  // PCR 23, reset allowed, App specific, extend all
};

#ifndef ARRAYSIZE
#  define ARRAYSIZE(a) (sizeof(a) / sizeof(a[0]))
#endif

MUST_BE(ARRAYSIZE(s_initAttributes) == IMPLEMENTATION_PCR);

#if ALG_SHA256 != YES && ALG_SHA384 != YES
#  error No default PCR banks defined
#endif

static const TPM_ALG_ID DefaultActivePcrBanks[] = {
#if ALG_SHA256
    TPM_ALG_SHA256
#endif
#if ALG_SHA384
#  if ALG_SHA256
    ,
#  endif
    TPM_ALG_SHA384
#endif
};

UINT32 _platPcr__NumberOfPcrs()
{
    return ARRAYSIZE(s_initAttributes);
}

// return the initialization attributes of a given PCR.
// pcrNumber expected to be in [0, _platPcr__NumberOfPcrs)
// returns the attributes for PCR[0] if the requested pcrNumber is out of range.
PCR_Attributes _platPcr__GetPcrInitializationAttributes(UINT32 pcrNumber)
{
    if(pcrNumber >= _platPcr__NumberOfPcrs())
    {
        pcrNumber = 0;
    }
    return s_initAttributes[pcrNumber];
}
```

```
// should the given PCR algorithm default to active in a new TPM?
BOOL _platPcr_IsPcrBankDefaultActive(TPM_ALG_ID pcrAlg)
{
    // brute force search is fast enough for a small array.
    for(int i = 0; i < ARRAYSIZE(DefaultActivePcrBanks); i++)
    {
        if(DefaultActivePcrBanks[i] == pcrAlg)
        {
            return TRUE;
        }
    }
    return FALSE;
}

// Fill a given buffer with the PCR initialization value for a particular PCR and hash
// combination, and return its length.  If the platform doesn't have a value, then
// the result size is expected to be zero, and the rfunction will return TPM_RC_PCR.
// If a valid is not available, then the core TPM library will ignore the value and
// treat it as non-existant and provide a default.
// If the buffer is not large enough for a pcr consistent with pcrAlg, then the
// platform will return TPM_RC_FAILURE.
TPM_RC _platPcr__GetInitialValueForPcr(
    UINT32     pcrNumber,        // IN: PCR to be initialized
    TPM_ALG_ID pcrAlg,           // IN: Algorithm of the PCR Bank being initialized
    BYTE       startupLocality,  // IN: locality where startup is being called from
    BYTE*      pcrData,          // OUT: buffer to put PCR initialization value into
    uint16_t   bufferSize,       // IN: maximum size of value buffer can hold
    uint16_t*  pcrLength  // OUT: size of initialization value returned in pcrBuffer
)
{
    // If the reset locality contains locality 4, then this
    // indicates a DRTM PCR where the reset value is all ones,
    // otherwise it is all zero.  Don't check with equal because
    // resetLocality is a bitfield of multiple values and does
    // not support extended localities.
    uint16_t pcrSize = CryptHashGetDigestSize(pcrAlg);
    pAssert_RC(pcrNumber < _platPcr__NumberOfPcrs());
    pAssert_RC(bufferSize >= pcrSize) pAssert_RC(pcrLength != NULL);

    PCR_Attributes pcrAttributes =
        _platPcr__GetPcrInitializationAttributes(pcrNumber);
    BYTE defaultValue = 0;
    // PCRs that can be cleared from locality 4 are DRTM and initialize to all 0xFF
    if((pcrAttributes.resetLocality & 0x10) != 0)
    {
        defaultValue = 0xFF;
    }
    MemorySet(pcrData, defaultValue, pcrSize);
    if(pcrNumber == HCRTM_PCR)
    {
        pcrData[pcrSize - 1] = startupLocality;
    }

    // platform could provide a value here if the platform has initialization rules
    // different from the original PC Client spec (the default used by the Core
library).
    *pcrLength = pcrSize;
    return TPM_RC_SUCCESS;
}
```

/Platform/src/PowerPlat.c

```
//** Includes and Function Prototypes

#include "Platform.h"
```

```
//** Functions

//***_plat__Signal_PowerOn()
// Signal platform power on
LIB_EXPORT int _plat__Signal_PowerOn(void)
{
    // Reset the timer
    _plat__TimerReset();

    // Need to indicate that we lost power
    s_powerLost = TRUE;

    return 0;
}

//*** _plat__WasPowerLost()
// Test whether power was lost before a _TPM_Init.
//
// This function will clear the "hardware" indication of power loss before return.
// This means that there can only be one spot in the TPM code where this value
// gets read. This method is used here as it is the most difficult to manage in the
// TPM code and, if the hardware actually works this way, it is hard to make it
// look like anything else. So, the burden is placed on the TPM code rather than the
// platform code
//  Return Type: int
//      TRUE(1)         power was lost
//      FALSE(0)        power was not lost
LIB_EXPORT int _plat__WasPowerLost(void)
{
    int retVal  = s_powerLost;
    s_powerLost = FALSE;
    return retVal;
}

//*** _plat_Signal_Reset()
// This a TPM reset without a power loss.
LIB_EXPORT int _plat__Signal_Reset(void)
{
    // Initialize locality
    s_locality = 0;

    // Command cancel
    s_isCanceled = FALSE;

    _TPM_Init();

    // if we are doing reset but did not have a power failure, then we should
    // not need to reload NV ...

    return 0;
}

//***_plat__Signal_PowerOff()
// Signal platform power off
LIB_EXPORT void _plat__Signal_PowerOff(void)
{
    // Prepare NV memory for power off
    _plat__NVDisable((void*)FALSE, 0);

#if ACT_SUPPORT
    // Disable tick ACT tick processing
    _plat__ACT_EnableTicks(FALSE);
#endif

    return;
```

```
}
```

/Platform/src/PPPlat.c

```
//** Description

//    This module simulates the physical presence interface pins on the TPM.

//** Includes
#include "Platform.h"

//** Functions

//***_plat__PhysicalPresenceAsserted()
// Check if physical presence is signaled
//  Return Type: int
//      TRUE(1)         if physical presence is signaled
//      FALSE(0)        if physical presence is not signaled
LIB_EXPORT int _plat__PhysicalPresenceAsserted(void)
{
    // Do not know how to check physical presence without real hardware.
    // so always return TRUE;
    return s_physicalPresence;
}

//***_plat__Signal_PhysicalPresenceOn()
// Signal physical presence on
LIB_EXPORT void _plat__Signal_PhysicalPresenceOn(void)
{
    s_physicalPresence = TRUE;
    return;
}

//***_plat__Signal_PhysicalPresenceOff()
// Signal physical presence off
LIB_EXPORT void _plat__Signal_PhysicalPresenceOff(void)
{
    s_physicalPresence = FALSE;
    return;
}
```

/Platform/src/RunCommand.c

```
//**Introduction
// This module provides the platform specific entry and fail processing. The
// _plat__RunCommand() function is used to call to ExecuteCommand() in the TPM code.
// This function does whatever processing is necessary to set up the platform
// in anticipation of the call to the TPM including settup for error processing.
//
// The _plat__Fail() function is called when there is a failure in the TPM. The TPM
// code will have set the flag to indicate that the TPM is in failure mode.
// This call will then recursively call ExecuteCommand in order to build the
// failure mode response. When ExecuteCommand() returns to _plat__Fail(), the
// platform will do some platform specific operation to return to the environment in
// which the TPM is executing. For a simulator, setjmp/longjmp is used. For an OS,
// a system exit to the OS would be appropriate.

//** Includes and locals
#include "Platform.h"
#include <assert.h>
#include <setjmp.h>
#include <stdio.h>

jmp_buf s_jumpBuffer;
```

```
// The following extern globals are copied here from Global.h to avoid including all
of Tpm.h here.
// TODO: Improve the interface by which these values are shared.
extern BOOL g_inFailureMode;  // Indicates that the TPM is in failure mode
#if ALLOW_FORCE_FAILURE_MODE
extern BOOL g_forceFailureMode;  // flag to force failure mode during test
#endif
#if FAIL_TRACE
// The name of the function that triggered failure mode.
extern const char* s_failFunctionName;
#endif  // FAIL_TRACE
extern UINT32 s_failFunction;
extern UINT32 s_failLine;
extern UINT32 s_failCode;

//** Functions

//***_plat__RunCommand()
// This version of RunCommand will set up a jum_buf and call ExecuteCommand(). If
// the command executes without failing, it will return and RunCommand will return.
// If there is a failure in the command, then _plat__Fail() is called and it will
// longjump back to RunCommand which will call ExecuteCommand again. However, this
// time, the TPM will be in failure mode so ExecuteCommand will simply build
// a failure response and return.
LIB_EXPORT void _plat__RunCommand(
    uint32_t        requestSize,   // IN: command buffer size
    unsigned char*  request,       // IN: command buffer
    uint32_t*       responseSize,  // IN/OUT: response buffer size
    unsigned char** response       // IN/OUT: response buffer
)
{
    setjmp(s_jumpBuffer);
    ExecuteCommand(requestSize, request, responseSize, response);
}

//***_plat__Fail()
// This is the platform depended failure exit for the TPM.
LIB_EXPORT NORETURN void _plat__Fail(void)
{

#if ALLOW_FORCE_FAILURE_MODE
    // The simulator asserts during unexpected (i.e., un-forced) failure modes.
    if(!g_forceFailureMode)
    {
        fprintf(stderr, "Unexpected failure mode (code %d) in ", s_failCode);
#  if FAIL_TRACE
        fprintf(stderr, "function '%s' (line %d)\n", s_failFunctionName, s_failLine);
#  else   // FAIL_TRACE
        fprintf(stderr, "location code 0x%0x\n", s_locationCode);
#  endif  // FAIL_TRACE
        assert(FALSE);
    }

    // Clear the forced-failure mode flag for next time.
    g_forceFailureMode = FALSE;
#endif  // ALLOW_FORCE_FAILURE_MODE

    longjmp(&s_jumpBuffer[0], 1);
}
```

/Platform/src/Unique.c

```
//** Introduction
// In some implementations of the TPM, the hardware can provide a secret
```

```
// value to the TPM. This secret value is statistically unique to the
// instance of the TPM. Typical uses of this value are to provide
// personalization to the random number generation and as a shared secret
// between the TPM and the manufacturer.

//** Includes
#include "Platform.h"

#if VENDOR_PERMANENT_AUTH_ENABLED == YES

const char notReallyUnique[] = "This is not really a unique value. A real "
                               "unique value should"
                               " be generated by the platform.";

//** _plat__GetUnique()
// This function is used to access the platform-specific vendor unique values.
// This function places the unique value in the provided buffer ('b')
// and returns the number of bytes transferred. The function will not
// copy more data than 'bSize'.
// NOTE: If a platform unique value has unequal distribution of uniqueness
// and 'bSize' is smaller than the size of the unique value, the 'bSize'
// portion with the most uniqueness should be returned.
//
// 'which' indicates the unique value to return:
// 0 = RESERVED, do not use
// 1 = the VENDOR_PERMANENT_AUTH_HANDLE authorization value for this device
LIB_EXPORT uint32_t _plat__GetUnique(uint32_t which,  // which vendor value to return?
                                     uint32_t bSize,  // size of the buffer
                                     unsigned char* b  // output buffer
)
{
    const char* from   = notReallyUnique;
    uint32_t    retVal = 0;

    if(which == 1)
    {
        const size_t uSize =
            sizeof(notReallyUnique) <= bSize ? sizeof(notReallyUnique) : bSize;
        MemoryCopy(b, notReallyUnique, uSize);
    }
    // else fall through to default 0

    return retVal;
}

#endif
```

/Platform/src/VendorInfo.c

```
//** Introduction
// Provide vendor-specific version and identifiers to core TPM library for
// return in capabilities.  These may not be compile time constants and therefore
// are provided by platform callbacks.  These platform functions are expected to
// always be available, even in failure mode.
//
//** Includes
#include "Platform.h"

// In this sample platform, these are compile time constants, but are not required to
be.
#define MANUFACTURER    "XYZ "
#define VENDOR_STRING_1 "xCG "
#define VENDOR_STRING_2 "fTPM"
#define VENDOR_STRING_3 "\0\0\0\0"
#define VENDOR_STRING_4 "\0\0\0\0"
```

```
#define FIRMWARE_V1     (0x20240125)
#define FIRMWARE_V2     (0x00120000)
#define MAX_SVN         255

static uint32_t currentHash = FIRMWARE_V2;
static uint16_t currentSvn  = 10;

// Similar to the Core Library's ByteArrayToUint32, but usable in Platform code.
static uint32_t StringToUint32(char s[4])
{
    uint8_t* b = (uint8_t*)s;  // Avoid promotion to a signed integer type
    return (((uint32_t)b[0] << 8 | b[1]) << 8 | b[2]) << 8 | b[3];
}

// return the 4 character Manufacturer Capability code.  This
// should come from the platform library since that is provided by the manufacturer
LIB_EXPORT uint32_t _plat__GetManufacturerCapabilityCode()
{
    return StringToUint32(MANUFACTURER);
}

// return the 4 character VendorStrings for Capabilities.
// Index is ONE-BASED, and may be in the range [1,4] inclusive.
// Any other index returns all zeros. The return value will be interpreted
// as an array of 4 ASCII characters (with no null terminator)
LIB_EXPORT uint32_t _plat__GetVendorCapabilityCode(int index)
{
    switch(index)
    {
        case 1:
            return StringToUint32(VENDOR_STRING_1);
        case 2:
            return StringToUint32(VENDOR_STRING_2);
        case 3:
            return StringToUint32(VENDOR_STRING_3);
        case 4:
            return StringToUint32(VENDOR_STRING_4);
    }
    return 0;
}

// return the most-significant 32-bits of the TPM Firmware Version reported by
// getCapability.
LIB_EXPORT uint32_t _plat__GetTpmFirmwareVersionHigh()
{
    return FIRMWARE_V1;
}

// return the least-significant 32-bits of the TPM Firmware Version reported by
// getCapability.
LIB_EXPORT uint32_t _plat__GetTpmFirmwareVersionLow()
{
    return FIRMWARE_V2;
}

// return the TPM Firmware SVN reported by getCapability.
LIB_EXPORT uint16_t _plat__GetTpmFirmwareSvn(void)
{
    return currentSvn;
}

// return the TPM Firmware maximum SVN reported by getCapability.
LIB_EXPORT uint16_t _plat__GetTpmFirmwareMaxSvn(void)
{
    return MAX_SVN;
}
```

```
// Called by the simulator to set the TPM Firmware SVN reported by
// getCapability.
LIB_EXPORT void _plat__SetTpmFirmwareHash(uint32_t hash)
{
    currentHash = hash;
}

// Called by the simulator to set the TPM Firmware SVN reported by
// getCapability.
LIB_EXPORT void _plat__SetTpmFirmwareSvn(uint16_t svn)
{
    currentSvn = MIN(svn, MAX_SVN);
}

#if SVN_LIMITED_SUPPORT
// Dummy implmenentation for obtaining a Firmware SVN Secret bound
// to the given SVN.
LIB_EXPORT int _plat__GetTpmFirmwareSvnSecret(uint16_t  svn,
                                              uint16_t  secret_buf_size,
                                              uint8_t*  secret_buf,
                                              uint16_t* secret_size)
{
    int i;

    if(svn > currentSvn)
    {
        return -1;
    }

    // INSECURE dummy implementation: repeat the SVN into the secret buffer.
    for(i = 0; i < secret_buf_size; ++i)
    {
        secret_buf[i] = ((uint8_t*)&svn)[i % sizeof(svn)];
    }

    *secret_size = secret_buf_size;

    return 0;
}
#endif  // SVN_LIMITED_SUPPORT

#if FW_LIMITED_SUPPORT
// Dummy implmenentation for obtaining a Firmware Secret bound
// to the current firmware image.
LIB_EXPORT int _plat__GetTpmFirmwareSecret(
    uint16_t secret_buf_size, uint8_t* secret_buf, uint16_t* secret_size)
{
    int i;

    // INSECURE dummy implementation: repeat the firmware hash into the
    // secret buffer.
    for(i = 0; i < secret_buf_size; ++i)
    {
        secret_buf[i] = ((uint8_t*)&currentHash)[i % sizeof(currentHash)];
    }

    *secret_size = secret_buf_size;

    return 0;
}
#endif  // FW_LIMITED_SUPPORT

// return the TPM Type returned by TPM_PT_VENDOR_TPM_TYPE
LIB_EXPORT uint32_t _plat__GetTpmType()
{
```

```
    return 1;  // just the value the reference code has returned in the past.
}
```

Annex D
(informative)
Remote Procedure Interface

D.1 Introduction

These files provide an RPC interface for a TPM simulation.

The simulation uses two ports: a command port and a hardware simulation port. Only TPM commands defined in TPM 2.0 Part 3 are sent to the TPM on the command port. The hardware simulation port is used to simulate hardware events such as power on/off and locality; and indications such as _TPM_HashStart.

/Simulator/include/TpmTcpProtocol.h

```
//** Introduction

// TPM commands are communicated as uint8_t streams on a TCP connection.  The TPM
// command protocol is enveloped with the interface protocol described in this
// file. The command is indicated by a uint32 with one of the values below.  Most
// commands take no parameters return no TPM errors.  In these cases the TPM
// interface protocol acknowledges that command processing is completed by returning
// a uint32=0. The command TPM_SIGNAL_HASH_DATA takes a uint32-prepended variable
// length byte array and the interface protocol acknowledges command completion
// with a uint32=0. Most TPM commands are enveloped using the TPM_SEND_COMMAND
// interface command. The parameters are as indicated below.  The interface layer
// also appends a UIN32=0 to the TPM response for regularity.

//** Typedefs and Defines
#ifndef TCP_TPM_PROTOCOL_H
#define TCP_TPM_PROTOCOL_H

//** TPM Commands.
// All commands acknowledge processing by returning a uint32 == 0 except where noted
#define TPM_SIGNAL_POWER_ON      1
#define TPM_SIGNAL_POWER_OFF     2
#define TPM_SIGNAL_PHYS_PRES_ON  3
#define TPM_SIGNAL_PHYS_PRES_OFF 4
#define TPM_SIGNAL_HASH_START    5
#define TPM_SIGNAL_HASH_DATA     6
// {uint32_t BufferSize, uint8_t[BufferSize] Buffer}
#define TPM_SIGNAL_HASH_END 7
#define TPM_SEND_COMMAND    8
// {uint8_t Locality, uint32_t InBufferSize, uint8_t[InBufferSize] InBuffer} ->
//     {uint32_t OutBufferSize, uint8_t[OutBufferSize] OutBuffer}

#define TPM_SIGNAL_CANCEL_ON     9
#define TPM_SIGNAL_CANCEL_OFF    10
#define TPM_SIGNAL_NV_ON         11
#define TPM_SIGNAL_NV_OFF        12
#define TPM_SIGNAL_KEY_CACHE_ON  13
#define TPM_SIGNAL_KEY_CACHE_OFF 14

#define TPM_REMOTE_HANDSHAKE       15
#define TPM_SET_ALTERNATIVE_RESULT 16

#define TPM_SIGNAL_RESET   17
#define TPM_SIGNAL_RESTART 18

#define TPM_SESSION_END 20
#define TPM_STOP        21
```

```
#define TPM_GET_COMMAND_RESPONSE_SIZES 25

#define TPM_ACT_GET_SIGNALED 26

#define TPM_TEST_FAILURE_MODE 30

#define TPM_SET_FW_HASH 35
#define TPM_SET_FW_SVN  36

//** Enumerations and Structures
enum TpmEndPointInfo
{
    tpmPlatformAvailable = 0x01,
    tpmUsesTbs           = 0x02,
    tpmInRawMode         = 0x04,
    tpmSupportsPP        = 0x08
};

#ifdef _MSC_VER
#  pragma warning(push, 3)
#endif

// Existing RPC interface type definitions retained so that the implementation
// can be re-used
typedef struct in_buffer
{
    unsigned long  BufferSize;
    unsigned char* Buffer;
} _IN_BUFFER;

typedef unsigned char* _OUTPUT_BUFFER;

typedef struct out_buffer
{
    uint32_t       BufferSize;
    _OUTPUT_BUFFER Buffer;
} _OUT_BUFFER;

#ifdef _MSC_VER
#  pragma warning(pop)
#endif

#ifndef WIN32
typedef unsigned long DWORD;
typedef void*         LPVOID;
#endif

#endif
```

/Simulator/include/prototypes/Simulator_fp.h

```
/*(Auto-generated)
 *  Created by TpmPrototypes; Version 3.0 July 18, 2017
 *  Date: Mar  4, 2020  Time: 02:36:45PM
 */

#ifndef _SIMULATOR_FP_H_
#define _SIMULATOR_FP_H_

//** From TcpServer.c

#ifdef _MSC_VER
#elif defined(__unix__) || defined(__APPLE__)
#endif
```

```
//*** PlatformServer()
// This function processes incoming platform requests.
bool PlatformServer(SOCKET s);

//*** PlatformSvcRoutine()
// This function is called to set up the socket interfaces to listen for
// commands.
DWORD WINAPI PlatformSvcRoutine(LPVOID port);

//*** PlatformSignalService()
// This function starts a new thread waiting for platform signals.
// Platform signals are processed one at a time in the order in which they are
// received.
// If PickPorts is true, the server finds the next available port if the specified
// port was unavailable.
int PlatformSignalService(int PortNumber, bool PickPorts);

//*** RegularCommandService()
// This function services regular commands.
// If PickPorts is true, the server finds the next available port if the specified
// port was unavailable.
int RegularCommandService(int PortNumber, bool PickPorts);

//*** StartTcpServer()
// This is the main entry-point to the TCP server.  The server listens on the port
// specified.
// If PickPorts is true, the server finds the next available port if the specified
// port was unavailable.
//
// Note that there is no way to specify the network interface in this implementation.
int StartTcpServer(int PortNumber, bool PickPorts);

//*** ReadBytes()
// This function reads the indicated number of bytes ('NumBytes') into buffer
// from the indicated socket.
bool ReadBytes(SOCKET s, char* buffer, int NumBytes);

//*** WriteBytes()
// This function will send the indicated number of bytes ('NumBytes') to the
// indicated socket
bool WriteBytes(SOCKET s, char* buffer, int NumBytes);

//*** WriteUINT32()
// Send 4 byte integer
bool WriteUINT32(SOCKET s, uint32_t val);

//*** ReadUINT32()
// Function to read 4 byte integer from socket.
bool ReadUINT32(SOCKET s, uint32_t* val);

//*** ReadVarBytes()
// Get a uint32-length-prepended binary array.  Note that the 4-byte length is
// in network byte order (big-endian).
bool ReadVarBytes(SOCKET s, char* buffer, uint32_t* BytesReceived, int MaxLen);

//*** WriteVarBytes()
// Send a UINT32-length-prepended binary array.  Note that the 4-byte length is
// in network byte order (big-endian).
bool WriteVarBytes(SOCKET s, char* buffer, int BytesToSend);

//*** TpmServer()
// Processing incoming TPM command requests using the protocol / interface
// defined above.
bool TpmServer(SOCKET s);

//** From TPMCmdp.c
```

```
#ifdef _MSC_VER
#elif defined(__unix__) || defined(__APPLE__)
#endif

//*** Signal_PowerOn()
// This function processes a power-on indication. Among other things, it
// calls the _TPM_Init() handler.
void _rpc__Signal_PowerOn(bool isReset);

//*** Signal_Restart()
// This function processes the clock restart indication. All it does is call
// the platform function.
void _rpc__Signal_Restart(void);

//***Signal_PowerOff()
// This function processes the power off indication. Its primary function is
// to set a flag indicating that the next power on indication should cause
// _TPM_Init() to be called.
void _rpc__Signal_PowerOff(void);

//*** _rpc__ForceFailureMode()
// This function is used to debug the Failure Mode logic of the TPM. It will set
// a flag in the TPM code such that the next call to TPM2_SelfTest() will result
// in a failure, putting the TPM into Failure Mode.
void _rpc__ForceFailureMode(void);

//*** _rpc__Signal_PhysicalPresenceOn()
// This function is called to simulate activation of the physical presence "pin".
void _rpc__Signal_PhysicalPresenceOn(void);

//*** _rpc__Signal_PhysicalPresenceOff()
// This function is called to simulate deactivation of the physical presence "pin".
void _rpc__Signal_PhysicalPresenceOff(void);

//*** _rpc__Signal_Hash_Start()
// This function is called to simulate a _TPM_Hash_Start event. It will call
//
void _rpc__Signal_Hash_Start(void);

//*** _rpc__Signal_Hash_Data()
// This function is called to simulate a _TPM_Hash_Data event.
void _rpc__Signal_Hash_Data(_IN_BUFFER input);

//*** _rpc__Signal_HashEnd()
// This function is called to simulate a _TPM_Hash_End event.
void _rpc__Signal_HashEnd(void);

//*** _rpc__Send_Command()
// This is the interface to the TPM code.
//  Return Type: void
void _rpc__Send_Command(
    unsigned char locality, _IN_BUFFER request, _OUT_BUFFER* response);

//*** _rpc__Signal_CancelOn()
// This function is used to turn on the indication to cancel a command in process.
// An executing command is not interrupted. The command code may periodically check
// this indication to see if it should abort the current command processing and
// returned TPM_RC_CANCELLED.
void _rpc__Signal_CancelOn(void);

//*** _rpc__Signal_CancelOff()
// This function is used to turn off the indication to cancel a command in process.
void _rpc__Signal_CancelOff(void);

//*** _rpc__Signal_NvOn()
```

```
// In a system where the NV memory used by the TPM is not within the TPM, the
// NV may not always be available. This function turns on the indicator that
// indicates that NV is available.
void _rpc__Signal_NvOn(void);

//*** _rpc__Signal_NvOff()
// This function is used to set the indication that NV memory is no
// longer available.
void _rpc__Signal_NvOff(void);

//*** _rpc__RsaKeyCacheControl()
// This function is used to enable/disable the use of the RSA key cache during
// simulation.
void _rpc__RsaKeyCacheControl(int state);

//*** _rpc__ACT_GetSignaled()
// This function is used to count the ACT second tick.
bool _rpc__ACT_GetSignaled(uint32_t actHandle);

//*** _rpc__SetTpmFirmwareHash()
// This function is used to modify the firmware's hash during simulation.
void _rpc__SetTpmFirmwareHash(uint32_t hash);

//*** _rpc__SetTpmFirmwareSvn()
// This function is used to modify the firmware's SVN during simulation.
void _rpc__SetTpmFirmwareSvn(uint16_t svn);

//** From TPMCmds.c

//*** main()
// This is the main entry point for the simulator.
// It registers the interface and starts listening for clients
int main(int argc, char* argv[]);

#endif  // _SIMULATOR_FP_H_
```

/Simulator/src/simulatorPrivate.h

```
// common headers for simulator implementation files

#ifndef SIMULATOR_PRIVATE_H
#define SIMULATOR_PRIVATE_H

//** Includes, Locals, Defines and Function Prototypes
#include <public/tpm_public.h>

#include "simulator_sysheaders.h"

// TODO_RENAME_INC_FOLDER:prototypes refers to the platform library
#include <prototypes/platform_public_interface.h>
// TODO_RENAME_INC_FOLDER:platform_interface refers to the TPM_CoreLib platform
interface
#include <platform_interface/tpm_to_platform_interface.h>
#include <platform_interface/platform_to_tpm_interface.h>

#include "TpmTcpProtocol.h"
#include "Simulator_fp.h"

#endif  // SIMULATOR_PRIVATE_H
```

/Simulator/src/simulator_sysheaders.h

```
// system headers for the simulator, both Windows and Linux
```

```
#ifndef _SIMULATOR_SYSHEADERS_H_
#define _SIMULATOR_SYSHEADERS_H_
// include the system headers silencing warnings that occur with /Wall
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#ifdef _MSC_VER
#  pragma warning(push, 3)
// C4668 is supposed to be level 4, but this is still necessary to suppress the
// error.  We don't want to suppress it globally because the same error can
// happen in the TPM code and it shouldn't be ignored in those cases because it
// generally means a configuration header is missing.
//
// X is not defined as a preprocessor macro, assuming 0 for #if
#  pragma warning(disable : 4668)
#  include <windows.h>
#  include <winsock.h>
#  pragma warning(pop)
typedef int socklen_t;
#elif defined(__unix__) || defined(__APPLE__)
#  include <unistd.h>
#  include <errno.h>
#  include <netinet/in.h>
#  include <sys/socket.h>
#  include <pthread.h>
// simulate certain windows APIs
#  define ZeroMemory(ptr, sz) (memset((ptr), 0, (sz)))
#  define closesocket(x)      close(x)
#  define INVALID_SOCKET      (-1)
#  define SOCKET_ERROR        (-1)
#  define WSAGetLastError()   (errno)
#  define WSAEADDRINUSE       EADDRINUSE
#  define INT_PTR             intptr_t
typedef int SOCKET;
#  define _strcmpi            strcasecmp
#else
#  error "Unsupported platform."
#endif  // _MSC_VER
#endif  // _SIMULATOR_SYSHEADERS_H_
```

/Simulator/src/TcpServer.c

```
//** Description
//
// This file contains the socket interface to a TPM simulator.
//
//** Includes, Locals, Defines and Function Prototypes
#include "simulatorPrivate.h"
#include <string.h>

// To access key cache control in TPM
void RsaKeyCacheControl(int state);

#ifndef __IGNORE_STATE__

static uint32_t ServerVersion = 1;

#  define MAX_BUFFER 1048576
char InputBuffer[MAX_BUFFER];   //The input data buffer for the simulator.
char OutputBuffer[MAX_BUFFER];  //The output data buffer for the simulator.
```

```
struct
{
    uint32_t largestCommandSize;
    uint32_t largestCommand;
    uint32_t largestResponseSize;
    uint32_t largestResponse;
} CommandResponseSizes = {0};

#endif  // __IGNORE_STATE___

//** Functions

//*** CreateSocket()
// This function creates a socket listening on 'PortNumber'.
// If PickPorts is true, the server finds the next available port if the specified
// port was unavailable.
static int CreateSocket(
    int PortNumber, bool PickPorts, SOCKET* ListenSocket, int* ActualPort)
{
    struct sockaddr_in MyAddress;
    int                res;
//
// Initialize Winsock
#ifdef _MSC_VER
    WSADATA wsaData;
    res = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if(res != 0)
    {
        printf("WSAStartup failed with error: %d\n", res);
        return -1;
    }
#endif
    // create listening socket
    *ListenSocket = socket(PF_INET, SOCK_STREAM, 0);
    if(INVALID_SOCKET == *ListenSocket)
    {
        printf("Cannot create server listen socket.  Error is 0x%x\n",
               WSAGetLastError());
        return -1;
    }
    // bind the listening socket to the specified port
    ZeroMemory(&MyAddress, sizeof(MyAddress));
    MyAddress.sin_port   = htons((unsigned short)PortNumber);
    MyAddress.sin_family = AF_INET;

    res = bind(*ListenSocket, (struct sockaddr*)&MyAddress, sizeof(MyAddress));
    if(PickPorts)
    {
        while(res == SOCKET_ERROR && MyAddress.sin_port < UINT16_MAX)
        {
            // keep trying as long as the underlying error is that the port is already
in use
            if(WSAGetLastError() != WSAEADDRINUSE)
            {
                break;
            }
            MyAddress.sin_port++;
            res =
                bind(*ListenSocket, (struct sockaddr*)&MyAddress, sizeof(MyAddress));
        }
    }
    if(res == SOCKET_ERROR)
    {
        printf("Bind error.  Error is 0x%x\n", WSAGetLastError());
        return -1;
    }
```

```
    // listen/wait for server connections
    res = listen(*ListenSocket, 3);
    if(res == SOCKET_ERROR)
    {
        printf("Listen error.  Error is 0x%x\n", WSAGetLastError());
        return -1;
    }

    *ActualPort = ntohs(MyAddress.sin_port);
    return 0;
}

//*** PlatformServer()
// This function processes incoming platform requests.
bool PlatformServer(SOCKET s)
{
    bool     OK = true;
    uint32_t Command;
    //
    for(;;)
    {
        OK = ReadBytes(s, (char*)&Command, 4);
        // client disconnected (or other error).  We stop processing this client
        // and return to our caller who can stop the server or listen for another
        // connection.
        if(!OK)
            return true;
        Command = ntohl(Command);
        switch(Command)
        {
            case TPM_SIGNAL_POWER_ON:
                _rpc__Signal_PowerOn(false);
                break;
            case TPM_SIGNAL_POWER_OFF:
                _rpc__Signal_PowerOff();
                break;
            case TPM_SIGNAL_RESET:
                _rpc__Signal_PowerOn(true);
                break;
            case TPM_SIGNAL_RESTART:
                _rpc__Signal_Restart();
                break;
            case TPM_SIGNAL_PHYS_PRES_ON:
                _rpc__Signal_PhysicalPresenceOn();
                break;
            case TPM_SIGNAL_PHYS_PRES_OFF:
                _rpc__Signal_PhysicalPresenceOff();
                break;
            case TPM_SIGNAL_CANCEL_ON:
                _rpc__Signal_CancelOn();
                break;
            case TPM_SIGNAL_CANCEL_OFF:
                _rpc__Signal_CancelOff();
                break;
            case TPM_SIGNAL_NV_ON:
                _rpc__Signal_NvOn();
                break;
            case TPM_SIGNAL_NV_OFF:
                _rpc__Signal_NvOff();
                break;
            case TPM_SIGNAL_KEY_CACHE_ON:
                _rpc__RsaKeyCacheControl(true);
                break;
            case TPM_SIGNAL_KEY_CACHE_OFF:
                _rpc__RsaKeyCacheControl(false);
```

```
                break;
            case TPM_SESSION_END:
                // Client signaled end-of-session
                TpmEndSimulation();
                return true;
            case TPM_STOP:
                // Client requested the simulator to exit
                return false;
            case TPM_TEST_FAILURE_MODE:
                _rpc__ForceFailureMode();
                break;
            case TPM_GET_COMMAND_RESPONSE_SIZES:
                OK = WriteVarBytes(
                    s, (char*)&CommandResponseSizes, sizeof(CommandResponseSizes));
                memset(&CommandResponseSizes, 0, sizeof(CommandResponseSizes));
                if(!OK)
                    return true;
                break;
            case TPM_ACT_GET_SIGNALED:
            {
                uint32_t actHandle;
                OK = ReadUINT32(s, &actHandle);
                WriteUINT32(s, _rpc__ACT_GetSignaled(actHandle));
                break;
            }
            case TPM_SET_FW_HASH:
            {
                uint32_t hash;
                OK = ReadUINT32(s, &hash);
                _rpc__SetTpmFirmwareHash(hash);
                break;
            }
            case TPM_SET_FW_SVN:
            {
                uint32_t svn;
                OK = ReadUINT32(s, &svn);
                _rpc__SetTpmFirmwareSvn((uint16_t)svn);
                break;
            }
            default:
                printf("Unrecognized platform interface command %d\n", (int)Command);
                WriteUINT32(s, 1);
                return true;
        }
        WriteUINT32(s, 0);
    }
}

//*** WritePortToFile()
// This function writes the given port out to a file.
bool WritePortToFile(const char* filename, int port)
{
    FILE* f;

#ifdef _MSC_VER
#  pragma warning(push)
#  pragma warning(disable : 4996)
#endif  // _MSC_VER
    f = fopen(filename, "w");
#ifdef _MSC_VER
#  pragma warning(pop)
#endif  // _MSC_VER
    if(f == NULL)
    {
        return false;
    }
```

```
    fprintf(f, "%d\n", port);
    return fclose(f) == 0;
}

//*** DeletePortFile()
// This function deletes the port file.
bool DeletePortFile(const char* filename)
{
    return remove(filename) == 0;
}

struct platformParameters
{
    int  port;
    bool pickPorts;
};

//*** PlatformSvcRoutine()
// This function is called to set up the socket interfaces to listen for
// commands.
DWORD WINAPI PlatformSvcRoutine(LPVOID parms)
{
    struct platformParameters* platformParms = (struct platformParameters*)parms;
    int                        PortNumber    = platformParms->port;
    bool                       PickPorts     = platformParms->pickPorts;
    SOCKET                     listenSocket, serverSocket;
    struct sockaddr_in         HerAddress;
    int                        res;
    socklen_t                  length;
    bool                       continueServing;
    const char*                portFile = "platform.port";

    res = CreateSocket(PortNumber, PickPorts, &listenSocket, &PortNumber);
    if(res != 0)
    {
        printf("Could not create platform service socket\n");
        return res;
    }
    if(!WritePortToFile(portFile, PortNumber))
    {
        printf("Could not write port to %s\n", portFile);
        return (DWORD)-1;
    }
    // Loop accepting connections one-by-one until we are killed or asked to stop
    // Note the platform service is single-threaded so we don't listen for a new
    // connection until the prior connection drops.
    do
    {
        printf("Platform server listening on port %d\n", PortNumber);

        // blocking accept
        length       = sizeof(HerAddress);
        serverSocket = accept(listenSocket, (struct sockaddr*)&HerAddress, &length);
        if(serverSocket == INVALID_SOCKET)
        {
            printf("Accept error.  Error is 0x%x\n", WSAGetLastError());
            return (DWORD)-1;
        }
        printf("Client accepted\n");

        // normal behavior on client disconnection is to wait for a new client
        // to connect
        continueServing = PlatformServer(serverSocket);
        closesocket(serverSocket);
    } while(continueServing);
```

```
    if(!DeletePortFile(portFile))
    {
        printf("Could not delete %s", portFile);
        return (DWORD)-1;
    }
    free(parms);
    return 0;
}

//*** PlatformSignalService()
// This function starts a new thread waiting for platform signals.
// Platform signals are processed one at a time in the order in which they are
// received.
// If PickPorts is true, the server finds the next available port if the specified
// port was unavailable.
int PlatformSignalService(int PortNumber, bool PickPorts)
{
    struct platformParameters* parms;

    parms = (struct platformParameters*)malloc(sizeof(struct platformParameters));
    parms->port      = PortNumber;
    parms->pickPorts = PickPorts;
#if defined(_MSC_VER)
    HANDLE hPlatformSvc;
    int    ThreadId;

    hPlatformSvc = CreateThread(NULL,
                                0,
                                (LPTHREAD_START_ROUTINE)PlatformSvcRoutine,
                                (LPVOID)parms,
                                0,
                                (LPDWORD)&ThreadId);
    if(hPlatformSvc == NULL)
    {
        printf("Could not create platform thread\n");
        return -1;
    }
    return 0;
#else
    pthread_t thread_id;
    int       ret;

    ret = pthread_create(&thread_id, NULL, (void*)PlatformSvcRoutine, (LPVOID)parms);
    if(ret == -1)
    {
        printf("Could not create platform thread: %s\n", strerror(ret));
    }
    return ret;
#endif  // _MSC_VER
}

//*** RegularCommandService()
// This function services regular commands.
// If PickPorts is true, the server finds the next available port if the specified
// port was unavailable.
int RegularCommandService(int PortNumber, bool PickPorts)
{
    SOCKET             listenSocket;
    SOCKET             serverSocket;
    struct sockaddr_in HerAddress;
    int                res;
    socklen_t          length;
    bool               continueServing;
    const char*        portFile = "command.port";

    res = CreateSocket(PortNumber, PickPorts, &listenSocket, &PortNumber);
```

```
    if(res != 0)
    {
        printf("Could not create command service socket\n");
        return res;
    }
    if(!WritePortToFile(portFile, PortNumber))
    {
        printf("Could not write port to %s\n", portFile);
        return -1;
    }
    // Loop accepting connections one-by-one until we are killed or asked to stop
    // Note the TPM command service is single-threaded so we don't listen for
    // a new connection until the prior connection drops.
    do
    {
        printf("TPM command server listening on port %d\n", PortNumber);

        // blocking accept
        length       = sizeof(HerAddress);
        serverSocket = accept(listenSocket, (struct sockaddr*)&HerAddress, &length);
        if(serverSocket == INVALID_SOCKET)
        {
            printf("Accept error.  Error is 0x%x\n", WSAGetLastError());
            return -1;
        }
        printf("Client accepted\n");

        // normal behavior on client disconnection is to wait for a new client
        // to connect
        continueServing = TpmServer(serverSocket);
        closesocket(serverSocket);
    } while(continueServing);

    if(!DeletePortFile(portFile))
    {
        printf("Could not delete %s", portFile);
        return -1;
    }
    return 0;
}

#if RH_ACT_0

//*** SimulatorTimeServiceRoutine()
// This function is called to service the time 'ticks'.
static unsigned long WINAPI SimulatorTimeServiceRoutine(LPVOID notUsed)
{
    // All time is in ms
    const int64_t tick     = 1000;
    uint64_t      prevTime = _plat__RealTime();
    int64_t       timeout  = tick;

    (void)notUsed;

    while(true)
    {
        uint64_t curTime;

#  if defined(_MSC_VER)
        Sleep((DWORD)timeout);
#  else
        struct timespec req = {timeout / 1000, (timeout % 1000) * 1000};
        struct timespec rem;
        nanosleep(&req, &rem);
#  endif  // _MSC_VER
        curTime = _plat__RealTime();
```

```
        // May need to issue several ticks if the Sleep() took longer than asked,
        // or no ticks at all, it Sleep() was interrupted prematurely.
        while(prevTime < curTime - tick / 2)
        {
            //printf("%05lld | %05lld\n",
            //      prevTime % 100000, (curTime - tick / 2) % 100000);
            _plat__ACT_Tick();
            prevTime += (uint64_t)tick;
        }
        // Adjust the next timeout to keep the average interval of one second
        timeout = tick + (prevTime - curTime);
        //prevTime = curTime;
        //printf("%04lld | c:%05lld | p:%05llu\n",
        //          timeout, curTime % 100000, prevTime);
    }
    return 0;
}

//*** ActTimeService()
// This function starts a new thread waiting to wait for time ticks.
// Return Type: int
//  ==0         success
//  !=0         failure
static int ActTimeService(void)
{
    static bool running = false;
    int         ret     = 0;
    if(!running)
    {
#  if defined(_MSC_VER)
        HANDLE hThr;
        int    ThreadId;
        //
        printf("Starting ACT thread...\n");
        //  Don't allow ticks to be processed before TPM is manufactured.
        _plat__ACT_EnableTicks(false);

        // Create service thread for ACT internal timer
        hThr = CreateThread(NULL,
                            0,
                            (LPTHREAD_START_ROUTINE)SimulatorTimeServiceRoutine,
                            (LPVOID)NULL,
                            0,
                            (LPDWORD)&ThreadId);
        if(hThr != NULL)
            CloseHandle(hThr);
        else
            ret = -1;
#  else
        pthread_t thread_id;
        //
        ret = pthread_create(
            &thread_id, NULL, (void*)SimulatorTimeServiceRoutine, (LPVOID)NULL);
#  endif  // _MSC_VER

        if(ret != 0)
            printf("ACT thread Creation failed\n");
        else
            running = true;
    }
    return ret;
}

#endif  // RH_ACT_0
```

```
//*** StartTcpServer()
// This is the main entry-point to the TCP server.  The server listens on the port
// specified.
// If PickPorts is true, the server finds the next available port if the specified
// port was unavailable.
//
// Note that there is no way to specify the network interface in this implementation.
int StartTcpServer(int PortNumber, bool PickPorts)
{
    int res;

#if ACT_SUPPORT
#  if !RH_ACT_0
#    error "Compliance tests currently require ACT_0 if ACT_SUPPORT"
#  endif
    // Start the Time Service routine
    res = ActTimeService();
    if(res != 0)
    {
        printf("TimeService failed\n");
        return res;
    }
#endif  // ACT_SUPPORT

    // Start Platform Signal Processing Service
    res = PlatformSignalService(PortNumber + 1, PickPorts);
    if(res != 0)
    {
        printf("PlatformSignalService failed\n");
        return res;
    }
    // Start Regular/DRTM TPM command service
    res = RegularCommandService(PortNumber, PickPorts);
    if(res != 0)
    {
        printf("RegularCommandService failed\n");
        return res;
    }
    return 0;
}

//*** ReadBytes()
// This function reads the indicated number of bytes ('NumBytes') into buffer
// from the indicated socket.
bool ReadBytes(SOCKET s, char* buffer, int NumBytes)
{
    int res;
    int numGot = 0;
    //
    while(numGot < NumBytes)
    {
        res = recv(s, buffer + numGot, NumBytes - numGot, 0);
        if(res == -1)
        {
            printf("Receive error.  Error is 0x%x\n", WSAGetLastError());
            return false;
        }
        if(res == 0)
        {
            return false;
        }
        numGot += res;
    }
    return true;
}
```

```
//*** WriteBytes()
// This function will send the indicated number of bytes ('NumBytes') to the
// indicated socket
bool WriteBytes(SOCKET s, char* buffer, int NumBytes)
{
    int res;
    int numSent = 0;
    //
    while(numSent < NumBytes)
    {
        res = send(s, buffer + numSent, NumBytes - numSent, 0);
        if(res == -1)
        {
            if(WSAGetLastError() == 0x2745)
            {
                printf("Client disconnected\n");
            }
            else
            {
                printf("Send error.  Error is 0x%x\n", WSAGetLastError());
            }
            return false;
        }
        numSent += res;
    }
    return true;
}

//*** WriteUINT32()
// Send 4 byte integer
bool WriteUINT32(SOCKET s, uint32_t val)
{
    uint32_t netVal = htonl(val);
    //
    return WriteBytes(s, (char*)&netVal, 4);
}

//*** ReadUINT32()
// Function to read 4 byte integer from socket.
bool ReadUINT32(SOCKET s, uint32_t* val)
{
    uint32_t netVal;
    //
    if(!ReadBytes(s, (char*)&netVal, 4))
        return false;
    *val = ntohl(netVal);
    return true;
}

//*** ReadVarBytes()
// Get a uint32-length-prepended binary array.  Note that the 4-byte length is
// in network byte order (big-endian).
bool ReadVarBytes(SOCKET s, char* buffer, uint32_t* BytesReceived, int MaxLen)
{
    int  length;
    bool res;
    //
    res = ReadBytes(s, (char*)&length, 4);
    if(!res)
        return res;
    length         = ntohl(length);
    *BytesReceived = length;
    if(length > MaxLen)
    {
        printf("Buffer too big.  Client says %d\n", length);
        return false;
```

```
    }
    if(length == 0)
        return true;
    res = ReadBytes(s, buffer, length);
    if(!res)
        return res;
    return true;
}

//*** WriteVarBytes()
// Send a uint32-length-prepended binary array.  Note that the 4-byte length is
// in network byte order (big-endian).
bool WriteVarBytes(SOCKET s, char* buffer, int BytesToSend)
{
    uint32_t netLength = htonl(BytesToSend);
    bool     res;
    //
    res = WriteBytes(s, (char*)&netLength, 4);
    if(!res)
        return res;
    res = WriteBytes(s, buffer, BytesToSend);
    if(!res)
        return res;
    return true;
}

//*** TpmServer()
// Processing incoming TPM command requests using the protocol / interface
// defined above.
bool TpmServer(SOCKET s)
{
    uint32_t    length;
    uint32_t    Command;
    uint8_t     locality;
    bool        OK;
    int         result;
    int         clientVersion;
    _IN_BUFFER  InBuffer;
    _OUT_BUFFER OutBuffer;
    //
    for(;;)
    {
        OK = ReadBytes(s, (char*)&Command, 4);
        // client disconnected (or other error).  We stop processing this client
        // and return to our caller who can stop the server or listen for another
        // connection.
        if(!OK)
            return true;
        Command = ntohl(Command);
        switch(Command)
        {
            case TPM_SIGNAL_HASH_START:
                _rpc__Signal_Hash_Start();
                break;
            case TPM_SIGNAL_HASH_END:
                _rpc__Signal_HashEnd();
                break;
            case TPM_SIGNAL_HASH_DATA:
                OK = ReadVarBytes(s, InputBuffer, &length, MAX_BUFFER);
                if(!OK)
                    return true;
                InBuffer.Buffer     = (uint8_t*)InputBuffer;
                InBuffer.BufferSize = length;
                _rpc__Signal_Hash_Data(InBuffer);
                break;
            case TPM_SEND_COMMAND:
```

```
            OK = ReadBytes(s, (char*)&locality, 1);
            if(!OK)
                return true;
            OK = ReadVarBytes(s, InputBuffer, &length, MAX_BUFFER);
            if(!OK)
                return true;
            InBuffer.Buffer      = (uint8_t*)InputBuffer;
            InBuffer.BufferSize  = length;
            OutBuffer.BufferSize = MAX_BUFFER;
            OutBuffer.Buffer     = (_OUTPUT_BUFFER)OutputBuffer;
            // record the number of bytes in the command if it is the largest
            // we have seen so far.
            if(InBuffer.BufferSize > CommandResponseSizes.largestCommandSize)
            {
                CommandResponseSizes.largestCommandSize = InBuffer.BufferSize;
                memcpy(&CommandResponseSizes.largestCommand,
                       &InputBuffer[6],
                       sizeof(uint32_t));
            }
            _rpc__Send_Command(locality, InBuffer, &OutBuffer);
            // record the number of bytes in the response if it is the largest
            // we have seen so far.
            if(OutBuffer.BufferSize > CommandResponseSizes.largestResponseSize)
            {
                CommandResponseSizes.largestResponseSize = OutBuffer.BufferSize;
                memcpy(&CommandResponseSizes.largestResponse,
                       &OutputBuffer[6],
                       sizeof(uint32_t));
            }
            OK = WriteVarBytes(s, (char*)OutBuffer.Buffer, OutBuffer.BufferSize);
            if(!OK)
                return true;
            break;
        case TPM_REMOTE_HANDSHAKE:
            OK = ReadBytes(s, (char*)&clientVersion, 4);
            if(!OK)
                return true;
            if(clientVersion == 0)
            {
                printf("Unsupported client version (0).\n");
                return true;
            }
            OK &= WriteUINT32(s, ServerVersion);
            OK &= WriteUINT32(
                s, tpmInRawMode | tpmPlatformAvailable | tpmSupportsPP);
            break;
        case TPM_SET_ALTERNATIVE_RESULT:
            OK = ReadBytes(s, (char*)&result, 4);
            if(!OK)
                return true;
            // Alternative result is not applicable to the simulator.
            break;
        case TPM_SESSION_END:
            // Client signaled end-of-session
            return true;
        case TPM_STOP:
            // Client requested the simulator to exit
            return false;
        default:
            printf("Unrecognized TPM interface command %d\n", (int)Command);
            return true;
    }
    OK = WriteUINT32(s, 0);
    if(!OK)
        return true;
}
```

```
}
```

/Simulator/src/

```
//** Description
// This file contains the functions that process the commands received on the
// control port or the command port of the simulator. The control port is used
// to allow simulation of hardware events (such as, _TPM_Hash_Start) to test
// the simulated TPM's reaction to those events. This improves code coverage
// of the testing.

//** Includes and Data Definitions
#include "simulatorPrivate.h"

static bool s_isPowerOn = false;

//** Functions

//*** Signal_PowerOn()
// This function processes a power-on indication. Among other things, it
// calls the _TPM_Init() handler.
void _rpc__Signal_PowerOn(bool isReset)
{
    // if power is on and this is not a call to do TPM reset then return
    if(s_isPowerOn && !isReset)
        return;
    // If this is a reset but power is not on, then return
    if(isReset && !s_isPowerOn)
        return;
    // Unless this is just a reset, pass power on signal to platform
    if(!isReset)
        _plat__Signal_PowerOn();
    // Power on and reset both lead to _TPM_Init()
    _plat__Signal_Reset();

    // Set state as power on
    s_isPowerOn = true;
}

//*** Signal_Restart()
// This function processes the clock restart indication. All it does is call
// the platform function.
void _rpc__Signal_Restart(void)
{
    _plat__TimerRestart();
}

//***Signal_PowerOff()
// This function processes the power off indication. Its primary function is
// to set a flag indicating that the next power on indication should cause
// _TPM_Init() to be called.
void _rpc__Signal_PowerOff(void)
{
    if(s_isPowerOn)
        // Pass power off signal to platform
        _plat__Signal_PowerOff();
    // This could be redundant, but...
    s_isPowerOn = false;

    return;
}

//*** _rpc__ForceFailureMode()
// This function is used to debug the Failure Mode logic of the TPM. It will set
// a flag in the TPM code such that the next call to TPM2_SelfTest() will result
```

```
// in a failure, putting the TPM into Failure Mode.
void _rpc__ForceFailureMode(void)
{
#if SIMULATION
    SetForceFailureMode();
#endif
    return;
}

//*** _rpc__Signal_PhysicalPresenceOn()
// This function is called to simulate activation of the physical presence "pin".
void _rpc__Signal_PhysicalPresenceOn(void)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... pass physical presence on to platform
        _plat__Signal_PhysicalPresenceOn();
    return;
}

//*** _rpc__Signal_PhysicalPresenceOff()
// This function is called to simulate deactivation of the physical presence "pin".
void _rpc__Signal_PhysicalPresenceOff(void)
{
    // If TPM is power on...
    if(s_isPowerOn)
        // ... pass physical presence off to platform
        _plat__Signal_PhysicalPresenceOff();
    return;
}

//*** _rpc__Signal_Hash_Start()
// This function is called to simulate a _TPM_Hash_Start event. It will call
//
void _rpc__Signal_Hash_Start(void)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... pass _TPM_Hash_Start signal to TPM
        _TPM_Hash_Start();
    return;
}

//*** _rpc__Signal_Hash_Data()
// This function is called to simulate a _TPM_Hash_Data event.
void _rpc__Signal_Hash_Data(_IN_BUFFER input)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... pass _TPM_Hash_Data signal to TPM
        _TPM_Hash_Data(input.BufferSize, input.Buffer);
    return;
}

//*** _rpc__Signal_HashEnd()
// This function is called to simulate a _TPM_Hash_End event.
void _rpc__Signal_HashEnd(void)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... pass _TPM_HashEnd signal to TPM
        _TPM_Hash_End();
    return;
}

//*** _rpc__Send_Command()
```

```
// This is the interface to the TPM code.
//  Return Type: void
void _rpc__Send_Command(
    unsigned char locality, _IN_BUFFER request, _OUT_BUFFER* response)
{
    // If TPM is power off, reject any commands.
    if(!s_isPowerOn)
    {
        response->BufferSize = 0;
        return;
    }
    // Set the locality of the command so that it doesn't change during the command
    _plat__LocalitySet(locality);
    // Do implementation-specific command dispatch
    _plat__RunCommand(
        request.BufferSize, request.Buffer, &response->BufferSize, &response->Buffer);
    return;
}

//*** _rpc__Signal_CancelOn()
// This function is used to turn on the indication to cancel a command in process.
// An executing command is not interrupted. The command code may periodically check
// this indication to see if it should abort the current command processing and
// returned TPM_RC_CANCELLED.
void _rpc__Signal_CancelOn(void)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... set the platform canceling flag.
        _plat__SetCancel();
    return;
}

//*** _rpc__Signal_CancelOff()
// This function is used to turn off the indication to cancel a command in process.
void _rpc__Signal_CancelOff(void)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... set the platform canceling flag.
        _plat__ClearCancel();
    return;
}

//*** _rpc__Signal_NvOn()
// In a system where the NV memory used by the TPM is not within the TPM, the
// NV may not always be available. This function turns on the indicator that
// indicates that NV is available.
void _rpc__Signal_NvOn(void)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... make the NV available
        _plat__SetNvAvail();
    return;
}

//*** _rpc__Signal_NvOff()
// This function is used to set the indication that NV memory is no
// longer available.
void _rpc__Signal_NvOff(void)
{
    // If TPM power is on...
    if(s_isPowerOn)
        // ... make NV not available
        _plat__ClearNvAvail();
```

```
    return;
}

void RsaKeyCacheControl(int state);

//*** _rpc__RsaKeyCacheControl()
// This function is used to enable/disable the use of the RSA key cache during
// simulation.
void _rpc__RsaKeyCacheControl(int state)
{
#if USE_RSA_KEY_CACHE
    RsaKeyCacheControl(state);
#else
    NOT_REFERENCED(state);
#endif
    return;
}

//*** _rpc__ACT_GetSignaled()
// This function is used to count the ACT second tick.
bool _rpc__ACT_GetSignaled(uint32_t actHandle)
{
#if ACT_SUPPORT
    // If TPM power is on...
    if(s_isPowerOn)
        // ... query the platform
        return _plat__ACT_GetSignaled(actHandle - TPM_RH_ACT_0);
#else   // ACT_SUPPORT
    NOT_REFERENCED(actHandle);
#endif  // ACT_SUPPORT
    return false;
}

//*** _rpc__SetTpmFirmwareHash()
// This function is used to modify the firmware's hash during simulation.
void _rpc__SetTpmFirmwareHash(uint32_t hash)
{
#if SIMULATION
    _plat__SetTpmFirmwareHash(hash);
#endif
}

//*** _rpc__SetTpmFirmwareSvn()
// This function is used to modify the firmware's SVN during simulation.
void _rpc__SetTpmFirmwareSvn(uint16_t svn)
{
#if SIMULATION
    _plat__SetTpmFirmwareSvn(svn);
#endif
}
```

/Simulator/src/

```
//** Description
// This file contains the entry point for the simulator.

//** Includes, Defines, Data Definitions, and Function Prototypes
#include "simulatorPrivate.h"
#include <CryptoInterface.h>

#define PURPOSE                                                             \
    "TPM 2.0 Reference Simulator.\n"                                        \
    "Copyright (c) Microsoft Corporation; Trusted Computing Group. All rights " \
    "reserved."
```

```
#define DEFAULT_TPM_PORT 2321

// Information about command line arguments (does not include program name)
static uint32_t     s_ArgsMask = 0;  // Bit mask of unmatched command line args
static int          s_Argc     = 0;
static const char** s_Argv     = NULL;

//** Functions

#if DEBUG
//*** Assert()
// This function implements a run-time assertion.
// Computation of its parameters must not result in any side effects, as these
// computations will be stripped from the release builds.
static void Assert(bool cond, const char* msg)
{
    if(cond)
        return;
    fputs(msg, stderr);
    exit(2);
}
#else
#  define Assert(cond, msg)
#endif

//*** Usage()
// This function prints the proper calling sequence for the simulator.
static void Usage(const char* programName)
{
    fprintf(stderr, "%s\n\n", PURPOSE);
    fprintf(stderr,
            "Usage:  %s [PortNum] [opts]\n\n"
            "Starts the TPM server listening on TCP port PortNum (by default "
            "%d).\n\n"
            "An option can be in the short form (one letter preceded with '-' or "
            "'/')\n"
            "or in the full form (preceded with '--' or no option marker at all).\n"
            "Possible options are:\n"
            "   -h (--help) or ? - print this message\n"
            "   -m (--manufacture) - forces NV state of the TPM simulator to be "
            "(re)manufactured\n"
            "   -p (--pick_ports) - choose the next available TCP ports "
            "automatically "
            "if PortNum is not available\n",
            programName,
            DEFAULT_TPM_PORT);
    exit(1);
}

//*** CmdLineParser_Init()
// This function initializes command line option parser.
static bool CmdLineParser_Init(int argc, char* argv[], int maxOpts)
{
    if(argc == 1)
        return false;

    if(maxOpts && (argc - 1) > maxOpts)
    {
        fprintf(stderr, "No more than %d options can be specified\n\n", maxOpts);
        Usage(argv[0]);
    }

    s_Argc     = argc - 1;
    s_Argv     = (const char**)(argv + 1);
    s_ArgsMask = (1 << s_Argc) - 1;
    return true;
```

```
}

//*** CmdLineParser_More()
// Returns true if there are unparsed options still.
static bool CmdLineParser_More(void)
{
    return s_ArgsMask != 0;
}

//*** CmdLineParser_IsOpt()
// This function determines if the given command line parameter represents a valid
// option.
static bool CmdLineParser_IsOpt(
    const char* opt,       // Command line parameter to check
    const char* optFull,   // Expected full name
    const char* optShort,  // Expected short (single letter) name
    bool        dashed     // The parameter is preceded by a single dash
)
{
    return 0 == strcmp(opt, optFull)
           || (optShort && opt[0] == optShort[0] && opt[1] == 0)
           || (dashed && opt[0] == '-' && 0 == strcmp(opt + 1, optFull));
}

//*** CmdLineParser_IsOptPresent()
// This function determines if the given command line parameter represents a valid
// option.
static bool CmdLineParser_IsOptPresent(const char* optFull, const char* optShort)
{
    int i;
    int curArgBit;
    Assert(s_Argv != NULL, "InitCmdLineOptParser(argc, argv) has not been invoked\n");
    Assert(optFull && optFull[0],
           "Full form of a command line option must be present.\n"
           "If only a short (single letter) form is supported, it must be"
           "specified as the full one.\n");
    Assert(!optShort || (optShort[0] && !optShort[1]),
           "If a short form of an option is specified, it must consist "
           "of a single letter only.\n");

    if(!CmdLineParser_More())
        return false;

    for(i = 0, curArgBit = 1; i < s_Argc; ++i, curArgBit <<= 1)
    {
        const char* opt = s_Argv[i];
        if((s_ArgsMask & curArgBit) && opt
           && (0 == strcmp(opt, optFull)
               || ((opt[0] == '/' || opt[0] == '-')
                   && CmdLineParser_IsOpt(
                       opt + 1, optFull, optShort, opt[0] == '-'))))
        {
            s_ArgsMask ^= curArgBit;
            return true;
        }
    }
    return false;
}

//*** CmdLineParser_Done()
// This function notifies the parser that no more options are needed.
static void CmdLineParser_Done(const char* programName)
{
    char delim = ':';
    int  i;
    int  curArgBit;
```

```
    if(!CmdLineParser_More())
        return;

    fprintf(stderr,
            "Command line contains unknown option%s",
            s_ArgsMask & (s_ArgsMask - 1) ? "s" : "");
    for(i = 0, curArgBit = 1; i < s_Argc; ++i, curArgBit <<= 1)
    {
        if(s_ArgsMask & curArgBit)
        {
            fprintf(stderr, "%c %s", delim, s_Argv[i]);
            delim = ',';
        }
    }
    fprintf(stderr, "\n\n");
    Usage(programName);
}

#if CRYPTO_LIB_REPORTING
void ReportCryptoLibs()
{
    _CRYPTO_IMPL_DESCRIPTION sym, hash, math = {0};
    _crypto_GetSymImpl(&sym);
    _crypto_GetHashImpl(&hash);
    _crypto_GetMathImpl(&math);
    printf("Crypto implementation information:\n");
    printf("  Symmetric:   %s (%s)\n", sym.name, sym.version);
    printf("  Hashing:     %s (%s)\n", hash.name, hash.version);
    printf("  Math:        %s (%s)\n", math.name, math.version);
}
#endif  // CRYPTO_LIB_REPORTING

//*** main()
// This is the main entry point for the simulator.
// It registers the interface and starts listening for clients
int main(int argc, char* argv[])
{
    bool manufacture = false;
    bool pick_ports  = false;
    int  PortNum     = DEFAULT_TPM_PORT;

    // Parse command line options

    if(CmdLineParser_Init(argc, argv, 2))
    {
        if(CmdLineParser_IsOptPresent("?", "?")
           || CmdLineParser_IsOptPresent("help", "h"))
        {
            Usage(argv[0]);
        }
        if(CmdLineParser_IsOptPresent("manufacture", "m"))
        {
            manufacture = true;
        }
        if(CmdLineParser_IsOptPresent("pick_ports", "p"))
        {
            pick_ports = true;
        }
        if(CmdLineParser_More())
        {
            int i;
            for(i = 0; i < s_Argc; ++i)
            {
                char* nptr    = NULL;
                int   portNum = (int)strtol(s_Argv[i], &nptr, 0);
```

```
                if(s_Argv[i] != nptr)
                {
                    // A numeric option is found
                    if(!*nptr && portNum > 0 && portNum < 65535)
                    {
                        PortNum = portNum;
                        s_ArgsMask ^= 1 << i;
                        break;
                    }
                    fprintf(stderr, "Invalid numeric option %s\n\n", s_Argv[i]);
                    Usage(argv[0]);
                }
            }
        }
        CmdLineParser_Done(argv[0]);
    }

#if CRYPTO_LIB_REPORTING
    ReportCryptoLibs();
#endif  // CRYPTO_LIB_REPORTING

    printf("LIBRARY_COMPATIBILITY_CHECK is %s\n",
           (LIBRARY_COMPATIBILITY_CHECK ? "ON" : "OFF"));
    // Enable NV memory
    _plat__NVEnable(NULL, 0);

    if(manufacture || _plat__NVNeedsManufacture())
    {
        printf("Manufacturing NV state...\n");
        if(TPM_Manufacture(MANUF_FIRST_TIME) != MANUF_OK)
        {
            // if the manufacture didn't work, then make sure that the NV file doesn't
            // survive. This prevents manufacturing failures from being ignored the
            // next time the code is run.
            _plat__NVDisable((void*)TRUE, 0);
            exit(1);
        }
        // Coverage test - repeated manufacturing attempt
        if(TPM_Manufacture(MANUF_REMANUFACTURE) != MANUF_ALREADY_DONE)
        {
            exit(2);
        }
        // Coverage test - re-manufacturing
        TPM_TearDown();
        if(TPM_Manufacture(MANUF_FIRST_TIME) != MANUF_OK)
        {
            exit(3);
        }
    }
    // Disable NV memory
    _plat__NVDisable((void*)FALSE, 0);

    StartTcpServer(PortNum, pick_ports);
    return EXIT_SUCCESS;
}
```