A PROOF-OF-CONCEPT IMPLEMENTATION OF EAP-TLS WITH TPM SUPPORT

Carolin Latze and Ulrich Ultes-Nitsche

University of Fribourg

{carolin.latze | uun}@unifr.ch
Boulevard de Pérolles 90
1700 Fribourg
Switzerland

ABSTRACT

Many people who have tried to configure their IEEE 802.11 enabled mobile phones to connect to a public wireless hotspot know one of the major differences between IEEE 802.11 networks and 2G: the missing standardized login process. While the 2G standard covers all aspects of the communication process, first IEEE 802.11 standards only targeted the data transmission. Due to this lack of standards for authentication, the login process and the missing secure subscriber identification, a number of different, mostly incompatible, login procedures have been established that are all far away from being as usable, comfortable and secure as 2G methods. This is why the authors of this paper propose to use EAP-TLS, which is a well established, secure and scalable authentication protocol, in combination with identities provided by a Trusted Platform Module (TPM) in order to archieve a high comfort for the user

This paper describes the concept, presents a Linux based implementation, and evaluates the approach in a testbed.

KEY WORDS

Security, Authentication Protocols, TPM, EAP-TLS

A PROOF-OF-CONCEPT IMPLEMENTATION OF EAP-TLS WITH TPM SUPPORT

1 INTRODUCTION

From the user's point of view, GSM networks are one of the simplest and most comfortable networks that exist. There is no need to configure anything, just buy a phone and Subscriber Identity Module (SIM) and use it. That is completely different for 802.11 networks as they do not provide a standardized authentication protocol. When GSM was released, it included an identity management and based on those identities one authentication protocol used in every GSM-enabled device. By contrast, at the time 802.11 was released, the standard concentrated on data transmission and not on identity management and authentication. All the authentication protocols that exist today came later and - as shown in [1] – are not comparable to GSM authentication. With the emergence of Trusted Platform Modules (TPMs), for the first time there exists a kind of integrated hardware token and identity provider in the world of computer networks comparable to the SIM card in GSM networks. In 2007, the authors of this paper proposed to use these TPMs with EAP-TLS to build a secure scalable and user-friendly authentication protocol for 802.11 networks, which is as comfortable as the GSM authentication protocol [1]. This paper presents a concrete realization of the proposed protocol, providing a proof-of-concept prototype of the protocol showing how easy it is to modify existing EAP-TLS implementations to use this new approach. Furthermore, the work shows, that this protocol may be used on every kind of 802.11 enabled device.

This paper is structured as follows: an introduction into EAP-TLS with TPM support is given, followed by a detailed description of the proof-of-concept prototype. The paper concludes with improvements for the presented prototype and conclusions.

2 EAP-TLS WITH TPM SUPPORT

In 2007, the authors of this work proposed to extend EAP-TLS with TPM support to implement a user-friendly, secure and scalable authentication protocol for 802.11 networks [1].

EAP-TLS refers to the integration of the Transport Layer Security (TLS) protocol within the Extensible Authentication Protocol (EAP). It is one of the most secure authentication protocols for 802.11 networks when using the *mutual* authentication mode. But using mutual authentication in EAP-TLS means that the client needs its own certificate, which is too complicated for naïve users. In 2002, the Trusted Computing Group (TCG) released the main specification of the new Trusted Platform Module (TPM) [2], which states that the TPM will come with a predefined certificate infrastructure explained in the next section. This TPM certification process relies on the fact, that TPMs may be uniquely identified. As such a certificate belongs to the TPM and not to the real person, it is possible to automate the certificate retrieval, which makes it much easier to use for naïve users. Because of this possibility to facilitate the certificate retrieval without compromising its security, the authors of this paper propose to use the TPM certificates in conjunction with EAP-TLS to implement a highly secure, scalable and user-friendly authentication protocol.

3 PROOF-OF-CONCEPT IMPLEMENTATION

The following sections show a proof-of-concept implementation of the authentication protocol presented above. It starts with an overview over the architecture and goes on with a detailed description of the components needed in this architecture.

3.1 Architecture Overview

In general, the architecture consists of three components: client, authentication server and Privacy CA as shown in *Figure 1*. The client is the one, which wants to be authenticated, the authentication server authenticates the client and the Privacy CA issues the certificates and provides a verification service. Such a Privacy CA will most likely be deployed by every

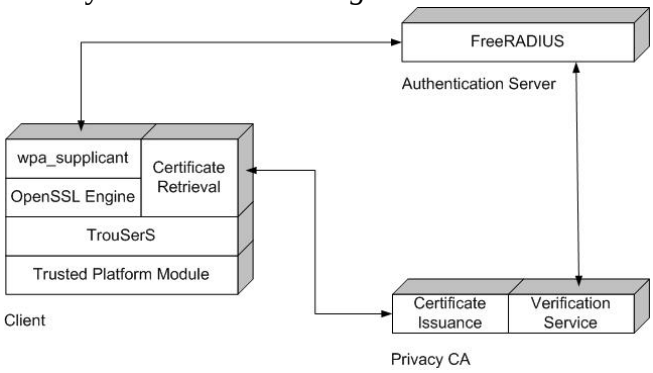

Figure 1 Architecture Overview

operator the enables EAP-TLS authentication using TPMs at its public wireless hotspots.

The client has to request a new identity before connecting to the EAP-TLS secured network. Afterwards, he may start an EAP-TLS authentication with the authentication server. During this EAP-TLS authentication process, the server has to verify the client's certificate. As TPM certificates are slightly different to X.509 certificates, there has to be a verification service, which does the job for the EAP-TLS authenticator.

3.1.1 The TPM

The Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG) [3] is a module, which amongst others, provides cryptographic functions and secure storage of keys and signatures. A very important feature of the TPM is that this module can be uniquely identified. It is equipped with a so called Endorsement Key Pair, which is unique. The private part of this key pair is never released from the TPM. The possibility to manage several identities (many different X.509 identity certificates) by the TPM makes the module very useful for different applications. The user might for instance use one identity for her e-banking account and the other one for an authentication as proposed in this paper. Using different identities for different purposes makes the user untraceable. Such a TPM identity must be signed by a certification authority (CA), which means that the CA is the only authority except the TPM's owner that is able to map the identities to a genuine TPM. To certify such a TPM identity, the CA has to check several certificates provided by the TPM manufacturer. That is why a special CA, called Privacy CA is needed to issue TPM identity certificates.

3.1.2 An Open Source TCG Software Stack: TrouSerS

The Trusted Platform Module (TPM) as described in [2] has to provide several cryptographic methods and protected storage. But it also has to be cheap to build to make it a ubiquitary device. That is why the TCG decided to distinguish between methods, that have to run in a protected environment and those that may run in a software-only environment called TCG Software Stack (TSS).

There are already some implementations of the TCG Software Stack, for instance the closed source NTRU TSS [4], an open source Java TSS

implementation called jTSS [5] and an open source IBM implementation called TrouSerS [6]. Given by the applications that had to be modified for the prototype, there was the constraint to use C as programming language and since TrouSerS is the only open source C implementation of the TSS, the authors decided to use it in their prototype implementation.

TrouSerS comes with a persistent storage file to store certain uncritical information on the hard disc in order to save memory on the TPM. This file contains for instance information about whether a key needs authentication or not (but it does not contain the authentication secret!) and may contain the public portion of keys, whose private key resides inside the TPM. To access those keys in the persistent storage, so called `UUID`s are used. There are certain predefined `UUID`s, for instance `{0, 0, 0, 0, 0, {0, 0, 0, 0, 0, 1}}` for the Storage Root Key (SRK), which is the parent for all keys stored in a given TPM.

3.1.3 The Privacy CA and TPM Certificates

As stated above, the TPM may manage different identities. This section describes the identity retrieval as specified in [2].

After having created a new RSA key that will serve as identity key, the TCG Software Stack (TSS) has to collect all information needed to request a new identity using `Tspi_TPM_CollateIdentityRequest`. This information includes: The endorsement credential, which identifies the TPM uniquely, the conformance credential, which states that the TPM is genuine, the platform credential, which states that the platform is genuine and the public portion of the newly generated identity key.

At the time of writing this paper, there exist only two implementations of such a Privacy CA: One in Java, implemented by the University of Graz [7] and freely available to install anywhere and another one available online [8]. The Java implementation uses the XML Key Management Protocol XKMS [9] for the communication between client and server, whereas the online PCA uses a REST-style AP [10]. The authors decided to use the latter one as this online PCA maps directly into client methods provided by TrouSerS.

After having verified all those certificates and public key information sent by the client, the PCA has to sign the identity certificate and send it back to the client. The client is now able to use this certificate. But identity

keys are special purpose keys, which cannot be directly used for TLS authentication. They are meant to be used to certify new keys, which may then be used for different purposes. Although the identity certificate itself is a valid X.509 certificate, those new certificates are not valid X.509 certificates anymore. The problem lies in the `basicContraint` extension of the identity certificate. According to the TCG, the extension has to be set to `CA:false` [2]. This means, that there is no possiblity to create a valid X.509 certificate beneath the identity certificate. The reason for this constraint lies in the structure of X.509 certificates:

```
Certificate ::= SEQUENCE {
        tbsCertificate        TBSCertificate,
        signatureAlgorithm    AlgorithmIdentifier,
        signatureValue        BIT STRING
}
TBSCertificate ::= SEQUENCE {
        version             EXPLICIT Version DEFAULT v1,
        serialNumber        CertificateSerialNumber,
        signature           AlgorithmIdentifier,
        issuer              Name,
        validity            Validity
        subject             SubjectPublicKeyInfo,
        issuerUniqueID      IMPLICIT UniqueIdentifier OPTIONAL,
        subjectUniqueID     IMPLICIT UniqueIdentifier OPTIONAL,
        extensions          EXPLICIT Extensions OPTIONAL
}
```

After having specified all the `TBSCertificate` values, the structure gets hashed and signed with the CA's key. The signature itself will be filled in to the `Certificate→signatureValue` field. The certified TPM keys are a bit different. The `TCPA_CERTIFY_INFO` structure, returned by the method used to certify keys (`Tspi_Key_CertifyKey`) looks like this:

```
typedef struct tdTPM_CERTIFY_INFO {
        TPM_STRUCT_VER              version;
        TPM_KEY_USAGE               keyUsage;
        TPM_KEY_FLAGS               keyFlags;
        TPM_AUTH_DATA_USAGE         authDataUsage;
        TPM_KEY_PARMS               algorithmParms;
        TPM_DIGEST                  pubkeyDigest;
        TPM_NONCE                   data;
        BOOL                        parentPCRStatus;
```

```
        UINT32                      PCRInfoSize;
        [size_is(pcrInfoSize)] BYTE* PCRInfo;
} TPM_CERTIFY_INFO;
```

This structure will also be hashed and signed by the identity key, but obviously, the values to be signed are completely different from those in X.509 certificates. However in order to use these so called certified keys in a TLS authentication, there has to be a possibility to transmit their public portion to the TLS server. As the TLS standard [11] requires X.509 certificates, the authors decided to implement the proof-of-concept prototype using slightly modified X.509 certificates in order to be able to integrate an unmodified TLS implementation. Therefore new X.509 extensions have been defined, which are simple aliases of the `nsComment` extension to transmit the `TCPA_CERTIFY_INFO` structure within an X.509 container. Those certificates cannot be verified by standard TLS implementations like OpenSSL [12]. In order to overcome this problem, the authors propose to use a verification service provided for instance by the PCA as explained below.

3.1.4 The Concept of an OpenSSL Engine

From OpenSSL version 0.9.6 on came a new concept called OpenSSL Engine [13]. Engine objects represent acceleration hardware or hardware tokens like smart cards to be used with OpenSSL. In 2007, IBM also provided an OpenSSL engine for TPMs [14]. This proof-of-concept prototype makes use of this OpenSSL TPM engine in order to integrate the TPM into the wpa_supplicant.

3.1.5 An Open Source EAP Peer – wpa_supplicant

There are several EAP peer implementations for every operating system like the open source XSupplicant [15] and wpa_supplicant [16] for Linux/Unix and Windows XP. Furthermore, there are also closed source application like the Cisco Secure Services Client [17] or the integrated Microsoft Windows client [18]. The authors decided to use wpa_supplicant, since this is the standard client in many Linux installations.

wpa_supplicant supports a wide range of actual wireless and wired authentication methods like WEP, WPA, WPA2 and many different EAP methods. For the cryptographic methods, it uses OpenSSL by default. It also

comes with engine support at least for smart cards. In order to enable also tpm engines, a new config option has been defined:

```
tpm_engine_path=/usr/local/lib/openssl/engines/libtpm.so
```

Furthermore, there has to be some tpm engine specific initialization code:

```
static int
tls_engine_load_dynamic_tpm(const char *tpm_so_path){
        char *engine_id = "tpm";
        const char *pre_cmd[] = {
                "SO_PATH", NULL /* tpm_so_path */,
                "ID", NULL /* engine_id */, "LIST_ADD", "1",
                "LOAD", NULL, NULL, NULL
        };
        if (!tpm_so_path) return 0;
        pre_cmd[1] = tpm_so_path;
        pre_cmd[3] = engine_id;
       wpa_printf(MSG_DEBUG, "ENGINE: Loading TPM Engine from
       %s",tpm_so_path);
return
tls_engine_load_dynamic_generic(pre_cmd, NULL, engine_id);
}
```

Basically that is all to integrate the OpenSSL TPM engine into wpa_supplicant. In order to use the TPM engine and certificates stored in the TPM, the client software has to specify the `tpm_engine_path`, `engine_id="tpm"`, the key's `UUID` as `key_id` and the TPM's owner password as `pin` in the wpa_supplicant's configuration file.

3.1.6 An Open Source EAP Authentication Server – FreeRADIUS

Similar to the different EAP supplicant implementations, there are also several implementations of authentication servers. There are comercial products from Cisco [19] or Microsoft [20], but in the open source community, the most widely deployed EAP authentication server is FreeRADIUS [21]. This is a modular authentication server, implemented in the C programming language. Authentication methods are implemented as modules, which makes it easily extensible.

As written above, the first prototype uses sligthly modified X.509 certificates, which allows using the EAP-TLS module with only minor changes. The only thing that needs to be changed is the certificate

verification as the TPM certificates are no valid X.509 certificates anymore. In order to be able to verify those certificates, the Privacy CA has to provide a verification service. The FreeRADIUS server tries to verify the client certificate as always using OpenSSL. In case OpenSSL cannot verify the certificate, the server has to open the certificate's extension to determine whether it is a TPM certificate or not. In case it came from a TPM, FreeRADIUS sends the TPM relevant X.509 extensions to the verification service, which replies with SUCCESS or FAILURE. Based on this reply, the FreeRADIUS server decides to authenticate the client or not. In order to avoid attacks on this verification process, the FreeRADIUS server and the verification service have to communicate over SSL using mutual authentication.

3.1.7 The Verification Service

In a valid setup, the client knows its certificate chain from root to its own certificate. In the proposed setup, the client's chain looks like this: `Privacy CA's root certificate → client's identity certificate → client certificate used for authentication`. As written above, the identity certificate comes with the `CA:false` constraint, which means, that this chain is not a valid X.509 chain. The valid part ends with the identity certificate. In a valid setup, the client sends its whole chain to the server, which is then able to verify the client's certificate easily, but in the prototype, the client will only send the last certificate and not the whole chain. But the Privacy CA (PCA) knows the whole chain! That means the establishement of a verification service at PCA solves the problem. In order for the verification service to be able to map the first part of the chain (`Privacy CA's root certificate → client's identity certificate`) to the client'certificate, it has to know the serial number of the approproate identity certificate. This number will be sent in the client's extensions. Using this number and the special TPM extensions, the PCA is able to verify the client's certificate.

4 IMPROVEMENTS

The usage of invalid X.509 certificates is probably not the best choice. Therefore, the next prototype will work with a new kind of certificates, designed for the special needs of the TPM:

```
Certificate ::= SEQUENCE {
```

```
        parentSerialNumber    CertificateSerialNumber,
        pubKey                OCTET STRING,
        tpmCertificate        TPMCertificate,
        signatureValue        BIT STRING
}
TPMCertificate ::= SEQUENCE {
        versionMajor                    OCTET,
        versionMinor                    OCTET,
        versionRevMajor                 OCTET,
        versionRevMinor                 OCTET,
        keyUsage                        OCTET STRING,
        keyFlags                        OCTET STRING,
        authDataUsage                   OCTET,
        algorithmID                     OCTET STRING,
        encScheme                       OCTET STRING,
        sigScheme                       OCTET STRING,
        parmSize                        INTEGER,
        parms                   [0]     OCTET STRING OPTIONAL,
                --If not present,parmSize MUST be 0--
        pubkeyDigest                    OCTET STRING,
        nonce                           OCTET STRING,
        parentPCRStatus                 BOOLEAN,
        PCRInfoSize             [1]     INTEGER OPTIONAL,
                --If not present,parentPCRStatus MUST be FALSE—
        PCRInfo                 [2]     OCTET STRING OPTIONAL,
                --If not present,parentPCRStatus MUST be FALSE—
}
```

Such a new certificate represents the `TPM_CERTIFY_INFO` structure perfectly. As this new certificate knows the serial number of its X.509 parent certificate, sending valid chains becomes possible again, even if those chains are no X.509 chains anymore! The new chain looks like this: `X.509 PCA Root Certificate → X.509 Identity Certificate → TPM Certificate` and will be stored in a file called `<name>.tpm`.

However, the Transport Layer Security protocol (TLS) requires X.509 certificates to work correctly [11], which means that a new protocol must be defined when using special TPM certificates. Therefore, the next version of the authentication protocol will be adapted and then called EAP-TPM.

5 EVALUATION AND CONCLUSION

The implementation has shown that existing EAP-TLS implementations may be adopted very easily in order to provide TPM support. On the client

side, the supplicant has to support TPM access to hold the private key in a secure environment. That is very similar to a smart card based setup, which is already supported by many supplicants. Things are bit more complicated on the server side. Due to the fact, that the certificates used in this implementation are invalid X.509 certificates, the server needs a new verification method. In order to avoid that the client has to send its identity certificate explicitly to the server, a verification service has been implemented, which is located at the Privacy CA. The Privacy CA already knows the identity certificates of its clients. The reason, why the authors decided not to send the identity certificate is the following: Using invalid X.509 certificates for the authentication is not desired for a productive version of the protocol. Those certificates are only used in this first prototype, which should show a proof-of-concept in a short time. Inserting a new message into the SSL handshake to transmit the identity certificate explicitly does not make sense since it had to be reverted for later versions. That is why it has been decided to use an external verification service instead. Furthermore, a new certificate type has been presented that is able to handle TPM certified keys and will be used in the next version of this new authentication protocol called EAP-TPM.

This prototype runs on a standard Linux system. The only applications that need to be modified are the supplicant and the RADIUS server, which means, that this prototype may run on every TPM equipped Linux based system, no matter whether it is a fully deployed computer or an embedded device.

Having such an authentication scheme will help to make 802.11 enabled mobile phones as popular as GSM phones. Furthermore, as this protocol runs also on normal computers, it will encourage more users to use public hotspots since it has never been so comfortable before.

6 REFERENCES

[1]**Latze, Carolin, Ultes-Nitsche, Ulrich und Baumgartner, Florian.** Strong Mutual Authentication in a User-Friendly Way in EAP-TLS. *Proceedings of the 15th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2007).* Split - Dubrovnik, Croatia : s.n., 2007.

[2]**Trusted Computing Platform Alliance (TCPA).** *Main Specification Version 1.1b.* 2002.

[3] The Trusted Computing Group. [Online] https://www.trustedcomputinggroup.org/home.
[4]NTRU - Products - Trusted Computing. [Online] http://www.ntru.com/products/tcg_ss.htm.
[5]**TU Graz.** Trusted Computing for the Java (tm) Platform. [Online] http://trustedjava.sourceforge.net.
[6] TrouSerS - The Open Source TCG Software Stack. [Online] http://trousers.sf.net.
[7]**TU Graz.** OpenTC PKI. [Online] http://opentc.iaik.tugraz.at.
[8]**Finney, Hal.** Privacy CA. [Online] http://www.privacyca.com.
[9]**W3C.** The XML Key Management Protocol. [Online] http://www.w2c.org/TR/xkms2.
[10]**Fielding, Roy Thomas.** *Architectural Styles and Design of Network-based Software Architectures.* s.l. : University of California, Irvine, 2000.
[11]**Dierks, T und Allen, C.** *The TLS Protocol - Version 1.0.* 1999. RFC 2246.
[12] OpenSSL. [Online] http://www.openssl.org.
[13]**Messier, Matt, Viega, John und Chandra, Pravir.** *Network Security with OpenSSL.* 2002.
[14] OpenSSL TPM Engine. [Online] 2007. http://sourceforge.net/project/showfiles.php?group_id=126012.
[15]IEEE 802.1X Open Source Implementation. [Online] http://open1x.sourceforge.net.
[16]Linux WPA/WPA2/IEEE 802.1X Supplicant. [Online] http://hostap.epitest.fi/wpa_supplicant/.
[17]Cisco Secure Services Client. [Online] http://www.cisco.com/en/US/prod/collateral/wireless/ps6442/ps7034/product_data_sheet0900aecd805081a7.html.
[18]Windows XP Wireless Auto Configuration. [Online] http://technet.microsoft.com/en-us/library/bb878124.aspx.
[19]Cisco Secure Access Control Server for Windows. [Online] http://www.cisco.com/en/US/products/sw/secursw/ps2086/index.html.
[20]Microsoft Internet Authentication Service. [Online] http://technet.microsoft.com/en-us/network/bb643123.aspx.
[21]The FreeRADIUS Project. [Online] http://www.freeradius.org.