Web Services Federation Language (WS-Federation)

Version 1.0
July 8 2003

Authors

Siddharth Bajaj, VeriSign
Giovanni Della-Libera, Microsoft
Brendan Dixon, Microsoft
Mike Dusche, Microsoft
Maryann Hondo, IBM
Matt Hur, Microsoft
Chris Kaler (Editor), Microsoft
Hal Lockhart, BEA
Hiroshi Maruyama, IBM
Anthony Nadalin (Editor), IBM
Nataraj Nagaratnam, IBM
Andrew Nash, RSA Security
Hemma Prafullchandra, VeriSign
John Shewchuk, Microsoft

Copyright Notice

Abstract

This specification defines mechanisms that are used to enable identity, account, attribute, authentication, and authorization federation across different trust realms.

Modular Architecture

By using the XML, SOAP and WSDL extensibility models, the WS* specifications are designed to be composed with each other to provide a rich Web services environment. WS-Federation by itself does not provide a complete security solution for Web services. WS-Federation is a building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models.

Status

This WS-Federation Specification is an initial public draft release and is provided for review and evaluation only. BEA, IBM, Microsoft, RSA Security and VeriSign hope to solicit your contributions and suggestions in the near future. BEA, IBM, Microsoft, RSA Security and VeriSign make no warrantees or representations regarding the specifications in any manner whatsoever.

Table of Contents

1. Introduction

This specification defines mechanisms to allow different security realms to federate using different or like mechanisms by allowing and brokering trust of identities, attributes, authentication between participating Web services.

The mechanisms defined in this specification can be used by passive and active requestors. The Web service requestors are assumed to understand the new security mechanisms and be capable of interacting with Web service providers.

This specification defines the model and framework for federation; subsequent documents define *profiles* which detail how different requestors apply this model.

1.1. Goals and Requirements

The primary goal of this specification is to enable federation of identity, attribute, authentication, and authorization information.

1.1.1 Requirements

The following list identifies the key driving requirements for this specification:

- Enable appropriate sharing of identity, authentication, and authorization data using different or like mechanisms
- Brokering of trust and security token exchange
- Local identities are not required at target services
- Optional hiding of identity information and other attributes

1.1.2. Non-Goals

The following topics are outside the scope of this document:

- Definition of message security or trust establishment/verification protocols
- Specification of new security token formats

- Specification of new attribute store interfaces
- Definition of security token assertion formats

1.2. Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

When describing abstract data models, this specification uses the notational convention used by the XML Infoset. Specifically, abstract property names always appear in square brackets (e.g., [some property]).

When describing concrete XML schemas, this specification uses the notational convention of WS-Security. Specifically, each member of an element's [children] or [attributes] property is described using an XPath-like notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute wildcard (<xs:anyAttribute/>).

1.3. Namespaces

The following namespaces are used in this document:

Prefix	Namespace
S	`http://www.w3.org/2002/06/soap-envelope`
wsse	`http://schemas.xmlsoap.org/ws/2003/07/secext`
wsu	`http://schemas.xmlsoap.org/ws/2002/07/utility`
wp	`http://schemas.xmlsoap.org/ws/2002/12/policy`

1.4. Schema and WSDL Files

The schema for this specification can be located at:

```
        http://schemas.xmlsoap.org/ws/2003/07/secext
```

The WSDL for this specification can be located at:

```
        http://schemas.xmlsoap.org/ws/2003/07/ws-federation.wsdl
```

1.5. Terminology

The following definitions establish the terminology and usage in this specification.

Passive Requestors – A *passive requestor* is an HTTP browser capable of broadly supported HTTP (e.g. HTTP/1.1).

Active Requestors – An *active requestor* is an application (possibly a Web browser) that is capable of issuing Web services messages such as those described in WS-Security and WS-Trust.

Profile – A *profile* is a document that describes how this model is applied to a specific class of requestor (e.g., passive, or active).

Claim – A *claim* is a declaration made by an entity (e.g. name, identity, key, group, privilege, capability, attribute, etc).

Security Token – A *security token* represents a collection of claims.

Signed Security Token – A *signed security token* is a security token that is asserted and cryptographically signed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket)

Proof-of-Possession – *Proof-of-possession* is authentication data that is provided with a message to prove that the message was sent and or created by a claimed identity.

Proof-of-Possession Token – A *proof-of-possession token* is a security token that contains data that a sending party can use to demonstrate proof-of-possession. Typically, although not exclusively, the proof-of-possession information is encrypted with a key known only to the sender and recipient.

Digest – A *digest* is a cryptographic checksum of an octet stream.

Signature - A *signature* is a value computed with a cryptographic algorithm and bound to data in such a way that intended recipients of the data can use the signature to verify that the data has not been altered since it was signed by the signer.

Security Token Service (STS) - A *security token service* is a Web service that issues security tokens (see WS-Security). That is, it makes assertions based on evidence that it trusts, to whoever trusts it. To communicate trust, a service requires proof, such as a security token or set of security tokens, and issues a security token with its own trust statement (note that for some security token formats this can just be a re-issuance or co-signature). This forms the basis of trust brokering.

Attribute Service - An *attribute service* is a Web service that maintains information (attributes) about principals within a trust realm or federation. The term principal, in this context, can be applied to any system entity, not just a person.

Pseudonym Service - A *pseudonym service* is a Web service that maintains alternate identity information about principals within a trust realm or federation. The term principal, in this context, can be applied to any system entity, not just a person.

Trust - *Trust is* the characteristic that one entity is willing to rely upon a second entity to execute a set of actions and/or to make set of assertions about a set of subjects and/or scopes.

Trust Domain/Realm - A *Trust Domain/Realm* is an administered security space in which the source and target of a request can determine and agree whether particular sets of credentials from a source satisfy the relevant security policies of the target. The target may defer the trust decision to a third party (if this has been established as part of the agreement) thus including the trusted third party in the Trust Realm.

Validation Service - A *validation service* is a Web service that uses the WS-Trust mechanisms to validate provided tokens and assess their level of trust (e.g. claims trusted).

Direct Trust – *Direct trust* is when a relying party accepts as true all (or some subset of) the claims in the token sent by the requestor.

Direct Brokered Trust – *Direct Brokered Trust* is when one party trusts a second party who, in turn, trusts or vouches for, the claims of a third party.

Indirect Brokered Trust – *Indirect Brokered Trust* is a variation on direct brokered trust where the second party can not immediately validate the claims of the third party to the first party and negotiates with the third party, or additional parties, to validate the claims and assess the trust of the third party.

Signature validation – *Signature validation* is the process of verifying that the message received is the same as the one sent.

Sender Authentication – *Sender authentication* is corroborated authentication evidence possibly across Web service actors/roles indicating the sender of a Web service message (and its associated data). Note that it is possible that a message may have multiple senders if authenticated intermediaries exist. Also note that it is application-dependent (and out of scope) as to how it is determined who first created the messages as the message originator might be independent of, or hidden behind an authenticated sender.

Realm or Domain – A *realm* or *domain* represents a single unit of security administration or trust.

Federation – A *federation* is a collection of realms that have established trust. The level of trust may vary, but typically includes authentication and may include authorization.

Identity Provider (IP) – *Identity Provider* is an entity that acts as an authentication service to end requestors and data origin authentication service to service providers (this is typically an extension of a security token service).

Single Sign On (SSO) – *Single Sign On* is an optimization of the authentication sequence to remove the burden of repeating actions placed on the requestor. To facilitate SSO, an element called an Identity Provider can act as a proxy on a requestor's behalf to provide evidence of authentication events to 3rd parties requesting information about the requestor. These Identity Providers (IP) are trusted 3rd parties and need to be trusted both by the requestor (to maintain the requestor's identity information as the loss of this information can result in the compromise of the requestors identity) and the Web services which may grant access to valuable resources and information based upon the integrity of the identity information provided by the IP.

Identity Mapping – *Identity Mapping* is a method of creating relationships between identity properties. Some Identity Providers may make use of identity mapping.

Sign-Out – A *sign-out* is the process by which a principal indicates that they will no longer be using their token and services in the realm can destroy their token caches for the principal.

Association – *Association* is the process by which principals become associated or affiliated with a trust realm or federation.

IP/STS – The acronym *IP/STS* is used to indicate a service that is either an identity provider (IP) or security token service (STS).

2. Model

This chapter describes the overall model for authentication which builds on the foundations specified in WS-Security, WS-Policy, and WS-Trust.

As described in WS-Trust, WS-PolicyAssertions can indicate that a Web Service requires a set of claims, codified in security tokens and related message elements, in order to process an incoming request. Upon evaluating the policy, if the requester does not have the necessary security token(s) to prove the required claims, it may use the mechanisms described in WS-Trust to use security tokens (or secrets) it has already to acquire additional security tokens.

This process of exchanging security tokens is typically bootstrapped by signing-on to obtain initial security tokens using either the mechanisms already defined in WS-Trust or the mechanisms defined in this specification. WS-PolicyExchange mechanisms are used to enable to the requestor to discover applicable policy, WSDL and schema about the endpoint.

These initial security tokens may be accepted by various Web services or exchanged at security token services (STS) / Identity Providers (IP) for additional security tokens subject to established trust relationships and trust policies as described in WS-Trust.

This specification also describes Attribute/Pseudonym services that can be utilized to provide mechanisms for restricted sharing of principal information and principal identity mapping (when different identities are used at different resources). WS-PolicyExchange mechanisms are used to enable to the requestor to discover the location of various Attribute/Pseudonym services.

Finally, it should be noted that just as a resource MAY act as its own IP/STS, a requestor MAY also act as its own IP/STS.

2.1. Trust and Security Token Issuance

The models defined in WS-Security, WS-Trust, and WS-Policy provides the basis for federation. This specification extends this foundation by describing how these models are combined to enable richer trust realm mechanisms across and within federations.

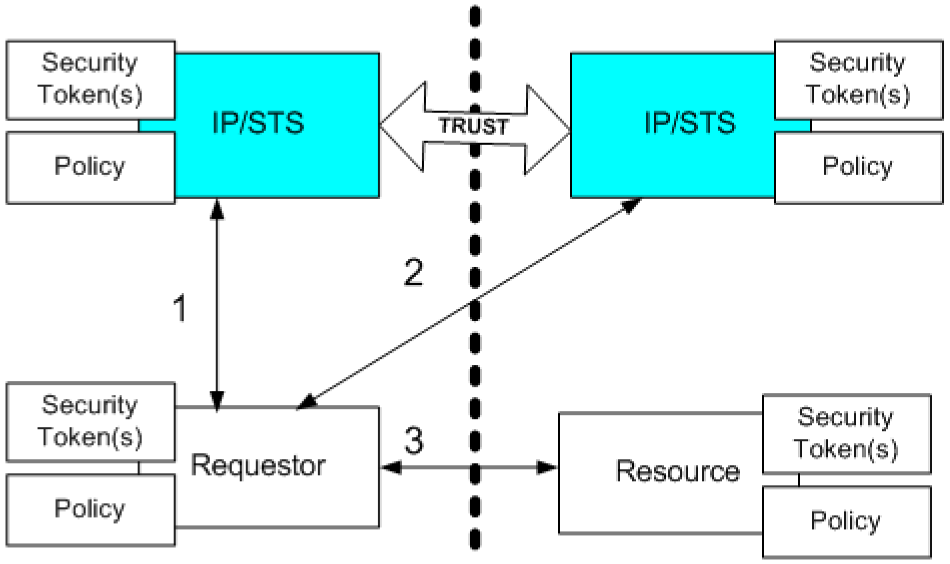

Figure 1 Basic STS

Figure 1 above illustrates one way the WS-Trust model may be applied to simple federation scenarios. Here security tokens (1) from the requestors trust realm are used to acquire security tokens from the resources trust realm (2) in order to access the resource/service (3). That is, a token from one STS is exchanged for another at a second STS or possibly stamped or cross-certified by a second STS).

Figure 2 below illustrates another approach where the resource/service uses a validation service. In this scenario, the resource provider (3) uses its security token service to understand and validate the security token(s) received from the requester (1,2). Note that the validity information is returned as a security token (since it includes authentication and/or authorization data).

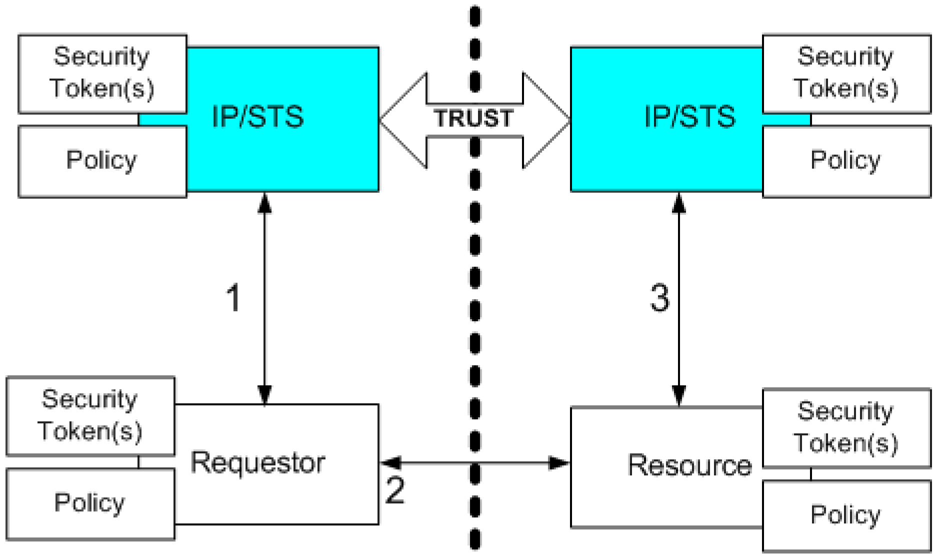

Figure 2 Alternate STS

In both of the above examples, a trust relationship has been established between the token services. Alternatively, as illustrated in Figure 3, there may not be a direct trust relationship, but an indirect trust relationship that relies on a third-party to establish and confirm trust.

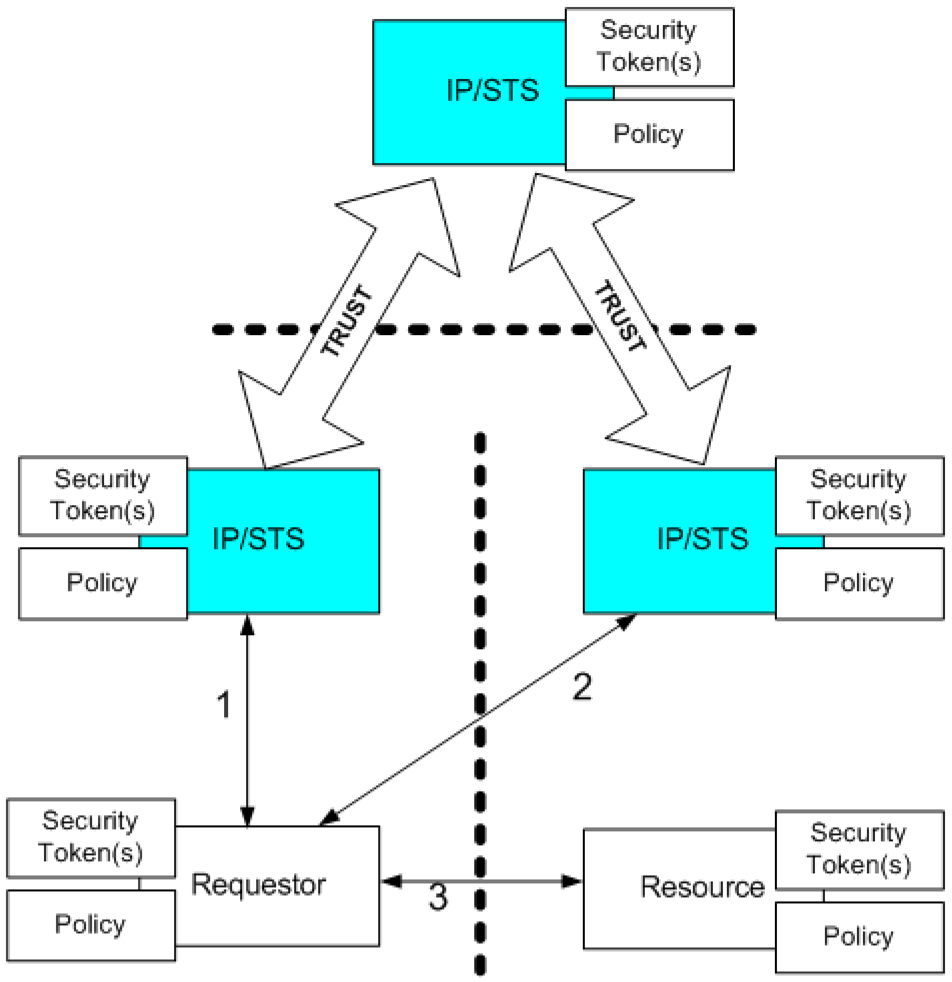

Figure 3 Indirect Trust

In practice, a requestor is likely to interact with multiple resources/services which are part of multiple trust realms as illustrated in the figure below:

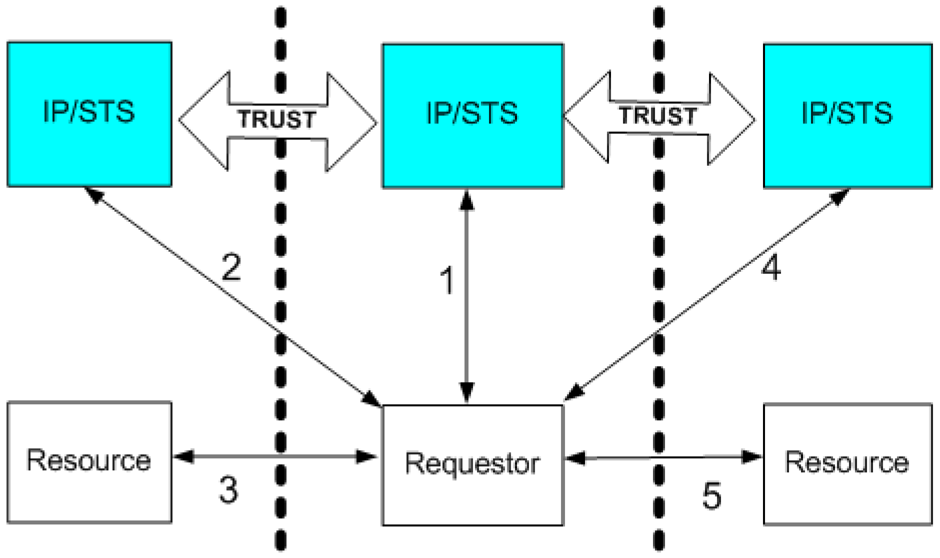

Figure 4 Multiple Trust Domains

Similarly, in response to a request a resource/service may need to access other resources/service on behalf of the requestor as illustrated in figure 5:

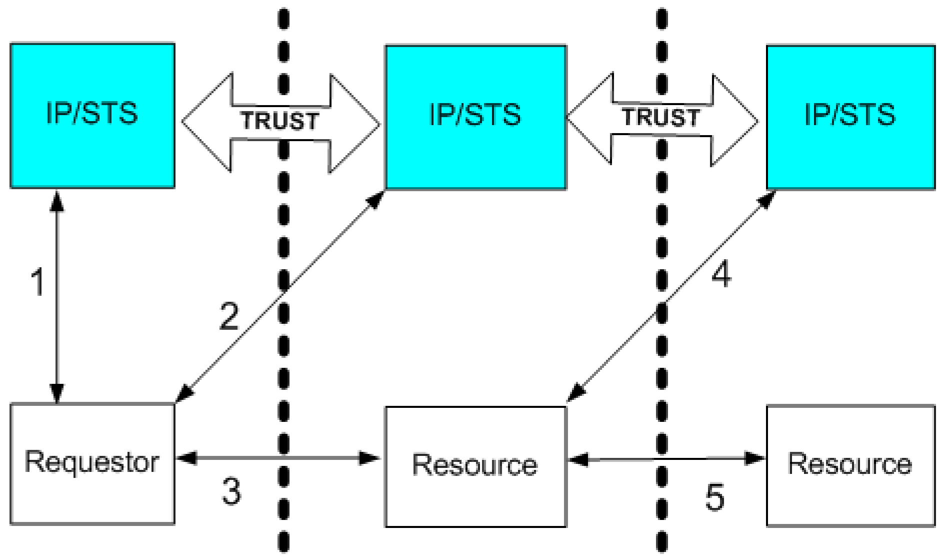

Figure 5 Delegation

In such cases (as illustrated in Figure 5) the (second) requestor provides security tokens to allow/indicate proof of delegation. It should be noted that there are a number of variations on this scenario. For example, the security token service for the final resource may only have a trust relationship with the token service from the original requestor (illustrated below), as opposed to the figure above where the trust doesn't exist with the original requestor's STS.

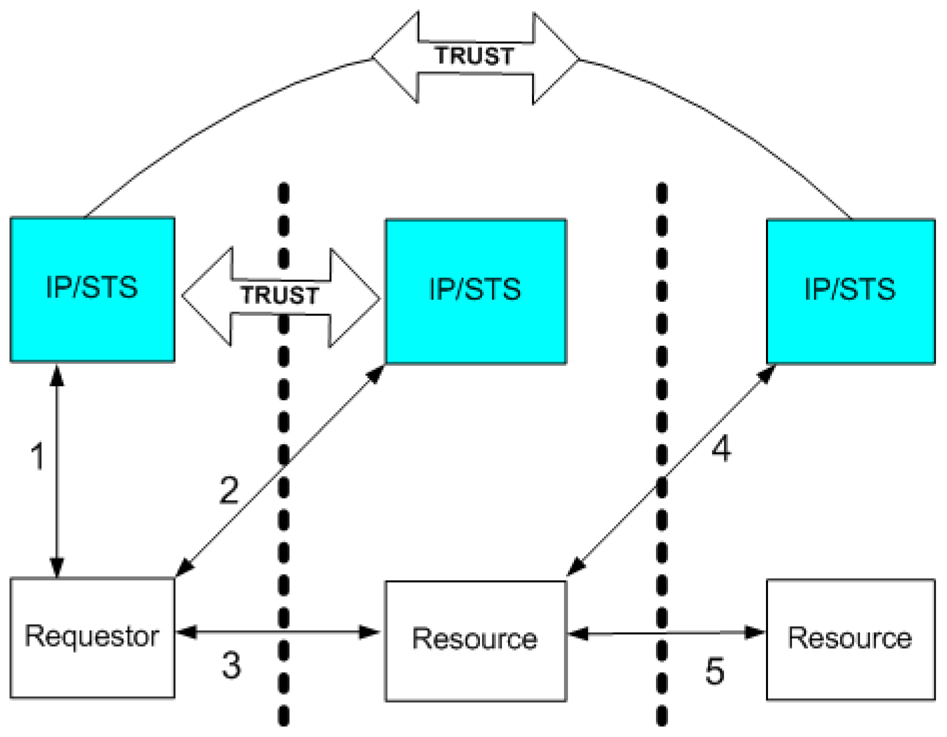

Figure 6 Delegation

Specifically, the resource or resource's security token service initiates a request for a security token that delegates the required claims. For more details on how to format such requests, refer to WS-Trust. These options are specified as part of the `<wsse:RequestSecurityToken>` request discussed in WS-Trust.

It should be noted that delegation tokens as well as the identity token of the delegation target need to be presented to the final service to ensure proper authorization.

In all cases, the original requestor's policy indicates the degree of delegation it is willing to support. Security token services SHOULD NOT allow any delegation or disclosure not specifically authorized by the original requestor, its policy, or by the service's policy.

2.2. Identity Providers

A security token service (STS) is a generic service that issues/exchanges security tokens using a common model and set of messages. As such, any Web service can, itself, be an STS simply by supporting the WS-Trust specification. Consequently, there are different types of security token services which provide different types of functions. For example, an STS might simply verify credentials for entrance to a realm or evaluate the trust of supplied security tokens.

Another function of a security token service is to provide identities – an *Identity Provider (IP)*. This is a special type of security token service that, at a minimum, performs peer entity authentication and can make identity claims in issued security tokens.

In many cases IP and STS services are interchangeable and many references within this document identify both.

The following example illustrates a possible combination of an IP and STS. In example 7, a requestor (1) obtains an identity security token from its IP and then presents/proves this to the STS for the desired resource. If successful (2), and if trust exists and authorization is approved, the STS returns an access token to the requestor. The requestor (3) then uses the access token on requests to the resource or Web service. Note that it is assumed that there is a trust relationship between the STS and the IP.

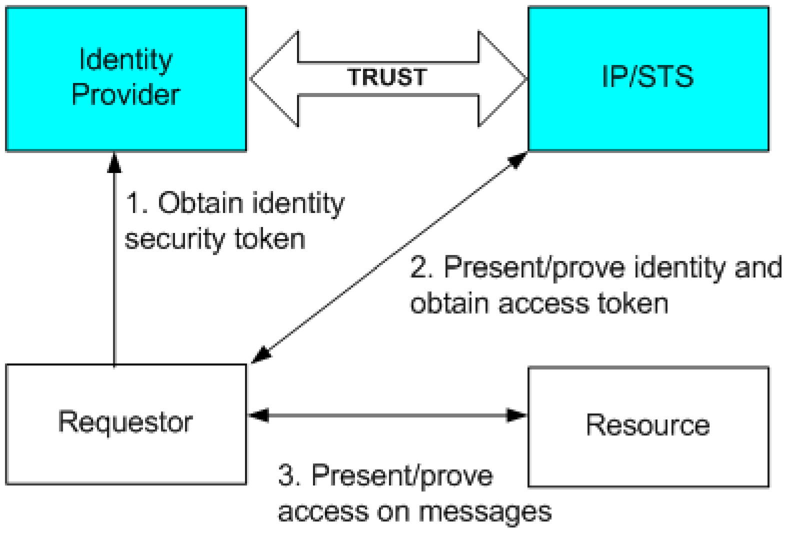
Identity Provider
TRUST
IP/STS
1. Obtain identity security token
2. Present/prove identity and obtain access token
Requestor
Resource
3. Present/prove access on messages

Figure 7 IP/STS

2.3. Attributes and Pseudonyms

Protecting privacy in a federated environment often requires additional controls and mechanisms. One such example is detailed access control on any information that may be considered personal or subject to privacy governances. Another example is obfuscation of identity information from identity providers (and security token services) to prevent unwanted correlation and to automatically map identities.

When requestors interact with resources in different trust realms (or different parts of a federation), there is often a need to *know* something about the requestor in order to personalize the experience. A service, known as an *attribute service* may be available within a realm or federation and such a service can be used to obtain authorized information about a principal. This allows the sharing of data between authorized entities.

To facilitate single sign-on where multiple identities need to be automatically mapped and the privacy of the identity need to maintained, there may also be a *pseudonym service*. A pseudonym service allows a principal to have different *aliases* at different resources/services or in different realms, and to optionally have the pseudonym change per-service or per-login. In some scenarios the identities are trusted when presented and in some cases identity mapping needs to occur from the identity to a pseudonym on behalf of the principal. The figure below illustrates two realms and their associated attribute/pseudonym services and some of the possible interactions. Note that it is assumed that there is a trust relationship between the realms.

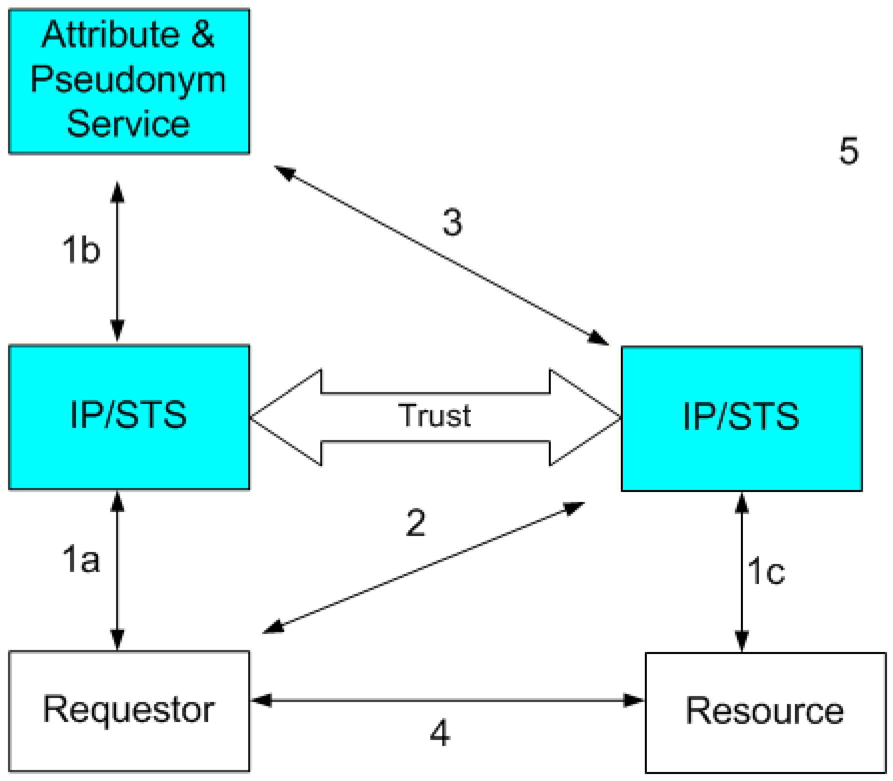

Figure 8 Attributes & Psedonyms

Figure 8 illustrates the general model for Atttribute & Pseudonym Services (note that there a different variations which are discussed later in this specification). In an initial (bootstrap) case, a requestor has knowledge of the policies of a resource, including its IP/STS. The requestor obtains its identity token from its IP/STS (1a) and communicates with the resource's IP/STS (2).

The resource IP/STS has registered a pseudonym with the requestor's pseudonym service (3). The requestor accesses the resource using the pseudonym token (4). The resource can obtain information (5) from the requestor's attribute service if authorized based on its identity token (1c). It should be noted that trust relationships will need to exist in order for the resource or its IP/STS to access the requestor's attribute or pseudonym service. In subsequent interactions, the requestor's IP/STS may automatically obtain pseudonym credentials for the resource (1b) if they are available. In such cases, steps 2 and 3 are omitted. Another possible scenario is that the requestor registers the tokens from step 2 with its pseudonym service directly (not illustrated).

Pseudonym services could be integrated with identity providers and security token services. Similarly, a pseudonym service could be integrated with an attribute service as a specialized form of attribute.

Pseudonyms are an optional mechanism that can be used by authorized cooperating services to federate identities and securely and safely access profile attribute information. This is done by allowing services to issue pseudonyms for authenticated identities and letting authorized services query for profile attributes which they are allowed to access, including pseudonyms specific to the requesting service.

While pseudonyms are helpful for principals who want to keep from having their activities tracked between the various sites they visit, they may add a level of complexity as the principal must typically manage the authorization and privacy of each pseudonym. For principals who find this difficult to coordinate, or don't have requirements that would necessitate pseudonyms, identity services MAY offer a constant identifier for that principal.

For example, a requestor authenticates with Business456.com with their primary identity "Fred.Jones". However, when the requestor interacts with Fabrikam123.com, he uses the pseudonym "Freddo".

Some identity services issue a constant identity marker such as a name or ID at a particular realm. However, there is often a desire to prevent identity tracking and collusion. This specification provides two possible countermeasures. The first is to have identity services issue random IDs each time a requestor signs in. This means that the resulting identity token the requester receives contains a unique identifier, typically random, that hides their identity. In this case it is "ABC123@Business456.com". The requestor then visits Fabrikam123.com. The Web service at Fabrikam123.com can request information about the requestor "ABC123@Business456.com" from the pseudonym/attribute service for Business456.com. If the requester has authorized it, the information will be provided by the identity service.

Alternatively, the requestor may indicate that they are "Freddo" to the Web service at Fabrikam123.com through some sort of mapping. Fabrikam123.com can now inform the pseudonym service for Business456.com that "ABC123@Business456.com" is known as "Freddo@Fabrikam123.com" (if authorized and allowed by the principal's privacy policy). This is illustrated below:

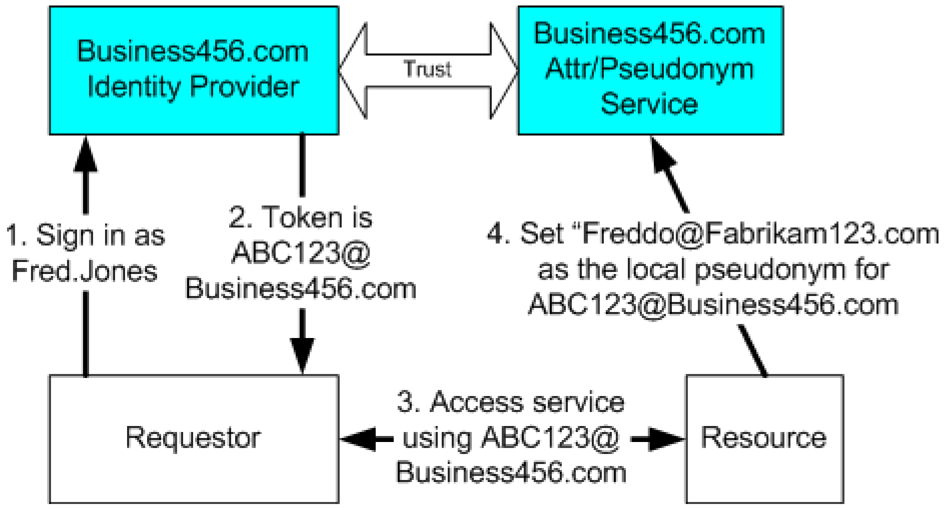
Business456.com Identity Provider
Trust
Business456.com Attr/Pseudonym Service
1. Sign in as Fred.Jones
2. Token is ABC123@ Business456.com
4. Set "Freddo@Fabrikam123.com as the local pseudonym for ABC123@Business456.com
Requestor
3. Access service using ABC123@ Business456.com
Resource

Figure 9 Pseudonym

Note that the attribute, pseudonym, and identity provider services could be combined or separated in many different configurations. Figure 9 illustrates a configuration where the IP is separate from the pseudonym service. In such a case there is shared information or specialized trust to allow the pseudonym service to perform the mapping or to make calls to the IP to facilitate the mapping. Different environments will have different configurations based on their needs, security policies, technologies used, and existing infrastructure.

The next time the requestor signs in to Business456.com Identity Provider, they might be given a new identifier like "XYZ321@Business456.com". This is possible because the pseudonym service interacts with the IP and is authorized and allowed under the principal's privacy policy to perform this action. Since Business456.com Identity Provider received the mapping, The Web service at Fabrikam123.com can now request a local pseudonym for XYZ321@Business456.com and be told "Freddo@Fabrikam123.com" (note that later in this section a mechanism for directly returning the pseudonym by the IP is discussed). The attribute service is able to do this because it has the ability to back-map "XYZ321@Business456.com" into a known identity at Business456.com which has associated with it pseudonyms for different realms. Figure 10 below illustrates this scenario:

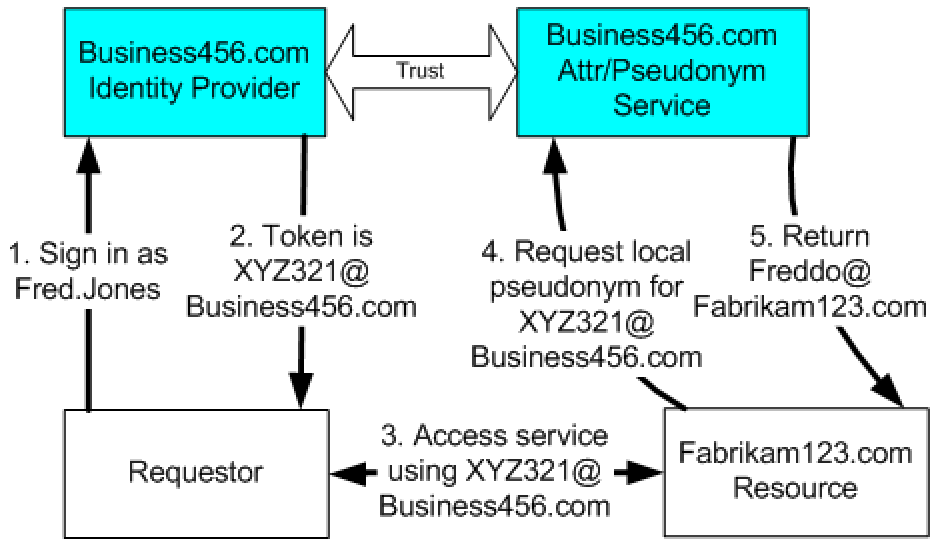
Business456.com Identity Provider
Trust
Business456.com Attr/Pseudonym Service
1. Sign in as Fred.Jones
2. Token is XYZ321@ Business456.com
4. Request local pseudonym for XYZ321@ Business456.com
5. Return Freddo@ Fabrikam123.com
Requestor
3. Access service using XYZ321@ Business456.com
Fabrikam123.com Resource

Figure 10 Pseudonym - local id

Again, this is possible because the pseudonym service interacts with the IP and is authorized and allowed under the principal's privacy policy to perform this action.

Now the service can map the given access name to a local name which can be used to obtain data stored within the local realm on behalf of the requestor as illustrated below:

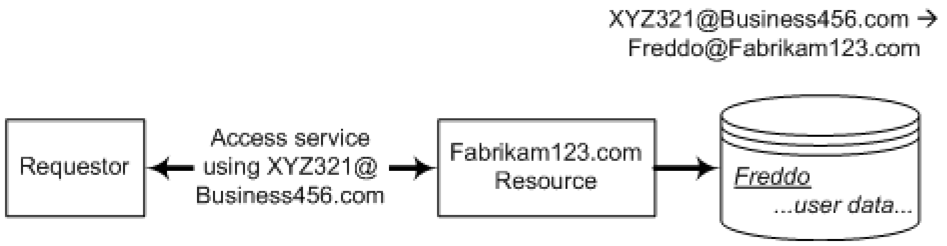
XYZ321@Business456.com → Freddo@Fabrikam123.com
Requestor
Access service using XYZ321@ Business456.com
Fabrikam123.com Resource
Freddo
...user data...

Figure 11 Pseudonym - local realm

Alternatively, the Identity Provider (or STS) can operate hand-in-hand with the pseudonym service. That is, the requestor asks its Identity Provider (or STS) for a token to a specified trust realm or resource/service. The STS looks for pseudonyms and issues a token which can be used at the specified resource/service as illustrated in figure 12 below:

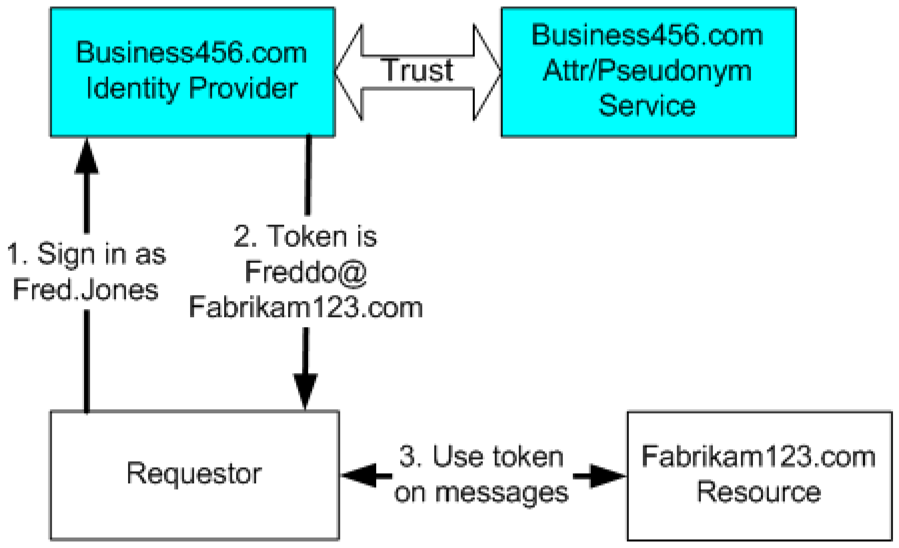
Business456.com Identity Provider
Trust
Business456.com Attr/Pseudonym Service
1. Sign in as Fred.Jones
2. Token is Freddo@ Fabrikam123.com
Requestor
3. Use token on messages
Fabrikam123.com Resource

Figure 11 Pseudonym – token acceptance

2.4. Summary

In summary, this specification extends the WS-Trust model to allow attributes and pseudonyms to be integrated into the token issuance mechanism to provide federated identity mapping mechanisms. Figure 13 below illustrates the key aspects of this new model:

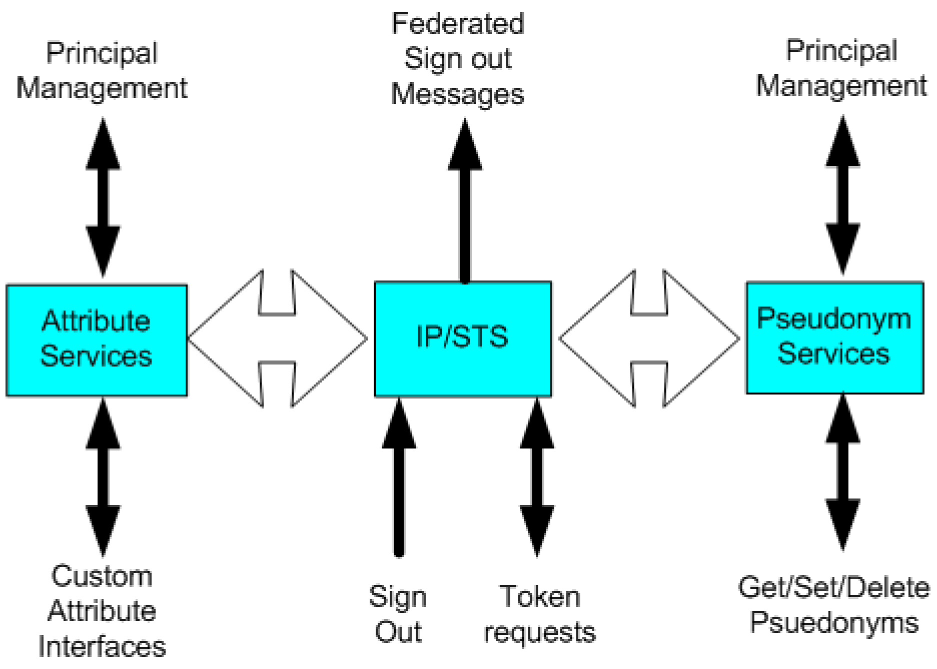

Figure 13 Pseudonym Issuance

As shown above Principals request security tokens from Identity Providers and security token services. As well, Principals MAY send sign-out requests (either explicitly or implicitly) indicating that cached or state information can be flushed immediately. Principals request tokens for resources/service using the mechanisms described in WS-Trust and the issued tokens may either represent the principals' primary identity or some pseudonym appropriate for the scope. The Identity Provider (or STS) sends messages to interested (and authorized) recipients. Principals are associated with the attribute/pseudonym services and attributes and pseudonyms are added and used.

In the sections that follow, additional details are provided on different aspects of this model.

3. Federation Metadata

Participation in a federation requires knowledge of metadata such as policies and potentially even WSDLs and schemas for the services within the federation. Additionally, in many cases mechanisms are needed to identify the Identity Provider, security token services, and attribute/pseudonym services for the target of a given policy (e.g. a Web service).

A variety of mechanisms MAY be used to acquire this metadata including:

1. The metadata is included in messages
2. Parties ask each other for the metadata
3. Previously exchanged metadata is remembered
4. Parties have preconfigured metadata, e.g. they "just know" possibly including in a "hard-coded" security token or policy.

3.1. WS-Policy and WS-MetadataExchange

To obtain and supply the information described above, this specification builds on the foundations outlined in WS-Policy and WS-MetadataExchange[1] to describe and acquire metadata. Readers should familiarize themselves with these specifications.

The mechanisms for the first three approaches above are defined in the WS-MetadataExchange specification.

3.2. Related Services

When a Web service describes its' policy, it MAY want to indicate where requestors can obtain security tokens in order to satisfy the services claims requirements.

To do this, this specification extends the mechanisms defined in WS-PolicyAssertions to define a `<wsse:RelatedService>` assertion.

The following table defines new service types for use with the `<wsse:RelatedService>` assertion:

QName	**Description**
`wsse:ServiceIP`	This is used to identify a related Identity Provider (IP).
`wsse:ServiceSTS`	This is used to identify a related security token service (STS).
`wsse:ServiceAS`	This is used to identify a related attribute service.
`wsse:ServicePS`	This is used to identify a related pseudonym service.

The following represents an overview of the syntax of the `<wsse:RelatedService>` element:

```
<wsse:RelatedService wsse:ServiceType="...">
    Endpoint-reference
</wsse:RelatedService>
```

The following describes elements and attributes used in a `<wsse:RelatedService>` element.

/RelatedService
: This assertion specificies a related service for the policy subject. The content (type) of this element is the same as an endpoint reference as defined in WS-Addressing.

/RelatedService/@wsse:ServiceType
: This required QName attribute indicates the type of service. The table above lists the pre-define service types, but XML namespaces can be used to define other service types.

The following example illustrates a policy using the `<wsp:RelatedService>` assertion:

```
<wsp:Policy>
    <wsse:RelatedService wsse:ServiceType="wsse:ServiceIP">
        Endpoint-reference
    </wsse:RelatedService>
    <wsse:RelatedService wsse:ServiceType="wsse:ServiceAS">
        Endpoint-reference
    </wsse:RelatedService>
    <wsse:RelatedService wsse:ServiceType="wsse:ServicePS">
        Endpoint-reference
    </wsse:RelatedService>
    ...
</wsp:Policy>
```

4. Sign-Out

The purpose of a *federated sign-out* is to clean up any cached state and security tokens that may exist within the federation. That is, sign-out notification serves as a hint that it is OK to flush cached data (such as security tokens) or state information for a specific principal. Depending on the type of application that is being used to sign-out, the process varies. For example, the implication of sign-out on currently active transactions is undefined and is resource-specific.

The exact mechanisms used for sign-out varies depending on the profile being used. In some cases, formal sign-out is implicit or not required. This section defines messages that MAY be used by profiles for explicit sign-out.

In general, sign-out messages are unreliable and correct operation must be ensured in their absence (i.e., the messages serve as hints only). Consequently, these messages must also be treated as idempotent since multiple deliveries could occur.

When sign-out is supported, it is typically provided as part of the IP/STS as it is usually the central processing point.

4.1. Sign-Out Message

The sign-out mechanism allows requestors to send a message to its IP/STS indicating that the requester is initiating a termination of the SSO. That is, cached information or state information can safely be flushed. This specification provides sign-out functions and it MAY be used, but the expected usage is that only token issuance and message security will be used.

For SOAP, the action of this message is as follows:

```
http://schemas.xmlsoap.org/2003/07/Federation#SignOut
```

The following represents an overview of the syntax of the `<SignOut>` element:

```
<wsse:SignOut wsu:Id="...">
    <wsse:Realm>...</wsse:Realm>
    <wsse:SignOutBasis>...<wsse:SignOutBasis>
</wsse:SignOut>
```

The `<wsse:SignOut>` message SHOULD be signed the requestor to prevent tampering and the signature SHOULD contain a timestamp to help prevent replay attacks.

The following describes elements and attributes used in a `<wsse:SignOut>` element.

/SignOut
    This element represents a sign-out message.

/SignOut/Realm
    This optional element specifies the "realm" to which the sign-out applies and is specified as an Endpoint Reference. If no realm is specified, then it is assumed that the recipient understands or assigns a fixed realm.

/SignOut/SignOutBasis
    The contents indicate the principal that is signing out. Note that any security token or security token reference MAY be used here and multiple tokens MAY be specified. That said, it is expected that the `<UsernameToken>` will be the most common. Note that a security token or security token reference MUST be specified.

/SignOut/@wsu:Id
    This optional attribute specifies a string label for this element.

/SignOut/@{any}
    This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/SignOut/{any}

This is an extensibility mechanism to allow additional elements to be used.

The <wsse:SignOut> message SHOULD be signed to prevent tampering and to prevent unauthorized sign-out messages (i.e., Alice sending a sign-out message for Bob without Bob's knowledge or permission). The signature SHOULD contain a timestamp to prevent replay attacks. it should be noted, however, that a principal MAY delegate the right to issue such messages on their behalf. The following represents an example of the <SignOut> message:

```
<S:Envelope>
    <S:Header>
        ...
        <wsu:Timestamp wsu:Id="ts">
            ...
        </wsu:Timestamp>
        <wsse:Security>
            <!-- Signature referecing IDs "ts" & "so" -->
            ...
        </wsse:Security>
    </S:Header>
    <S:Body>
        <wsse:SignOut wsu:Id="so">
          <wsse:SignOutBasis>
            <wsse:UsernameToken>
                <wsse:Username>NNK</wsse:Username>
            </wsse:UsernameToken>
          </wsse:SignOutBasis>
        </wsse:SignOut>
    </S:Body>
</S:Envelope>
```

4.2. Federating Sign-Out Messages

In many environments there is a need to take the messages indicating sign-out and distribute them across the federation, subject to authorization and privacy rules. Consequently, these messages result from when an explicit message is sent to the IP/STS (by either the principal or a delegate such as an IP/STS), or implicitly from an IP/STS as a result of some other action (such as a token request). In the case of the latter, an IP/STS indicates via its policy if such messages are automatically generated by observing the following WS-Policy assertion:

```
<wsse:AutoSignOutMessages/>
```

In order to request federated sign-out messages, interested parties (subject to principal and service authorization/privacy rules) use the `<RequestSSOMessages>` and `<CancelSSOMessages>` messages as described below. These messages are sent to the endpoint described in the policy for IP/STS as indicated in the following policy assertion (or to the default IP/STS endpoint the assertion is not present);

```
<wsse:RequestSSOMessagesEndpoint>
    ...endpoint reference as described in WS-Addressing...
</wsse:RequestSSOMessagesEndpoint>
```

The result of a successful request is that all compliant SSO messages generated implicitly or explicitly are sent to the requesting endpoints if allowed by the authorization/privacy rules.

The following illustrates the syntax of the `RequestSSOMessages` element:

```
<wsse:RequestSSOMessages>
    <wsa:EndpointReference>...</wsa:EndpointReference>
    <wsse:Realm>...</wsse:Realm>
    ...security tokens...
</wsse:RequestSSOMessages>
```

The following describes elements and attributes illustrated above:

/RequestSSOMessages

This element is used to request federated SSO messages.

/RequestSSOMessages/wsa:EndpointReference

This required element indicates the endpoint to which messages are to be sent.

/RequestSSOMessages/Realm

This optional element specifies the "realm" to which the sign-out applies and is specified as an Endpoint Reference. If no realm is specified, then it is assumed that the recipient understands or assigns a fixed realm.

/RequestSSOMessages/… security tokens(s) …

The contents of these optional elements restrict messages to only the specified identities. Note that any security token or security token reference MAY be used here and multiple tokens MAY be specified. That said, it is expected that the `<UsernameToken>` will be the most common.

/RequestSSOMessages/@{any}

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/RequestSSOMessages/{any}

This is an extensibility mechanism to allow additional elements to be used.

The following illustrates the syntax of the `CancelSSOMessages` element:

```
<wsse:CancelSSOMessages>
    <wsa:EndpointReference>...</wsa:EndpointReference>
    <wsse:Realm>...</wsse:Realm>
    ...security tokens...
```

```
</wsse:CancelSSOMessages>
```

The following describes elements and attributes illustrated above:

/CancelSSOMessages
: This element is used to remove requests for SSO messages.

/CancelSSOMessages/wsa:EndpointReference
: This required element indicates the endpoint to which message were to be sent.

/CancelSSOMessages/Realm
: This optional element specifies the "realm" to which the sign-out applies and is specified as an Endpoint Reference. If no realm is specified, then it is assumed that the recipient understands or assigns a fixed realm.

/CancelSSOMessages/... security tokens(s) ...
: The contents of these optional elements select which SSO messages to cancel based on the specified identities. Note that any security token or security token reference MAY be used here and multiple tokens MAY be specified. That said, it is expected that the `<UsernameToken>` will be the most common.

/CancelSSOMessages/@{any}
: This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element.

/CancelSSOMessages/{any}
: This is an extensibility mechanism to allow additional elements to be used.

The following example illustrates requesting SSO messages:

```
<wsse:RequestSSOMessages>
    <wsa:EndpointReference>
        <wsa:Reference>http://business456.com/SSO
             </wsa:Reference>
    </wsa:EndpointReference>
    <wsse:UsernameToken>
        <wsse:Username>Nicholas</wsse:Username>
    </wsse:UsernameToken>
</wsee:RequestSSOMessages>
```

5. Attribute Service

Web services need to be able to obtain authorized data related to authorized requestors to provide richer experiences. This can be addressed by having an attribute service that requesters and services may use to access restricted information (subject to authorization and privacy rules) where access is only granted to authorized services for any given attribute (or a separate service that has data available for such purposes).

Attribute stores and services most likely exist in some form already in service environments using service-specification protocols.

The figure below illustrates the conceptual namespace of an attribute service:

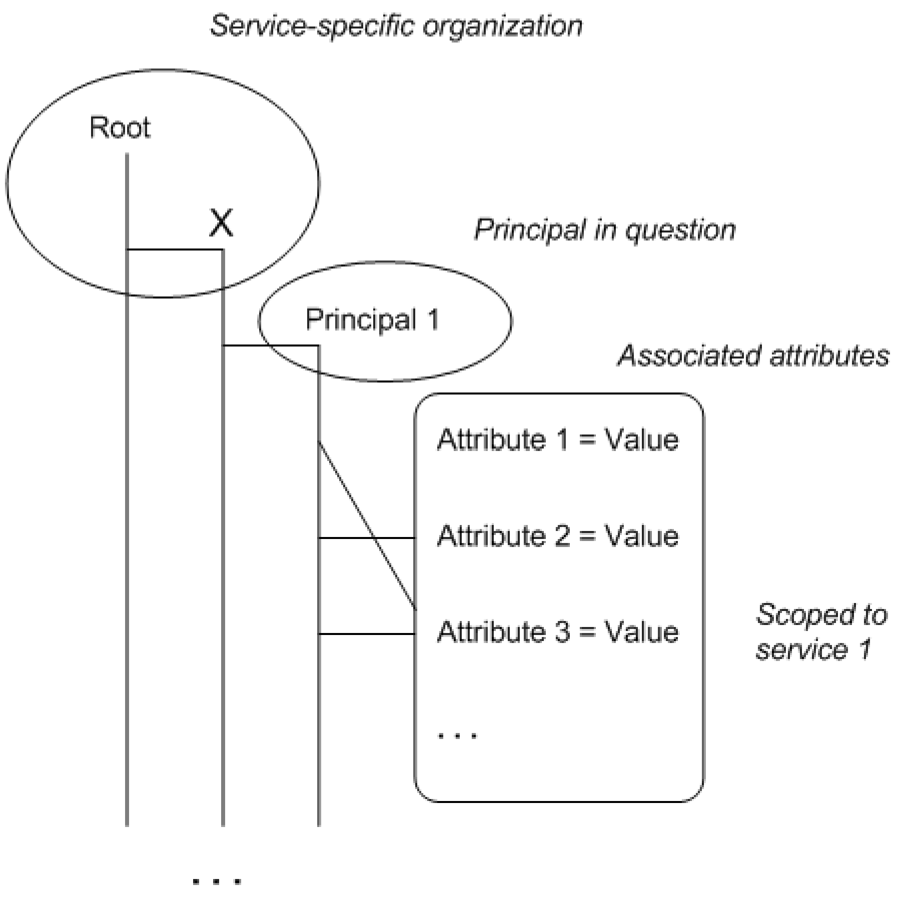

Figure 14 Attribute Service

An attribute service MAY leverage existing repositories and may provide some level of organization or context. That is, this specification makes no proposals or requirements on the organization of the data, just that a principal exist and be addressable using the mechanisms described here.

Principals represent any kind of resource, not just people. Consequently, the attribute mechanisms MAY be used to associate attributes with any resource, not just with identities. Said another way, principal identities represent just one class of resource that can be used by this specification.

A principal is one scope of attribute metadata to a particular subset of resources. That is, an attribute MAY have an associated scope to which it is applicable (possibly more than one resource/service), for example, nicknames used at specific sites. The scoping mechanism defined in WS-PolicyAttachment may be used to provide this capability.

It is expected that different attributes will be shared differently and require different degrees of privacy and protection. Consequently, each attribute expression SHOULD be capable of expressing its own access control and privacy policy. As well, the access control and privacy policy SHOULD take into account the associated scope(s) and principals that can speak for the scope(s).

Different services MAY support different types of attribute services which MAY be identified via policy[2].

Each attribute store may support different subsets of the functionality as described above. The store's policy indicates what functionality it supports.

5.1. UDDI as an Attribute Store

Services that are compliant with WS-Federation MAY also expose their attribute stores as UDDI endpoints. For UDDI to be used as an attribute store, one must know not only the endpoint address of the specific UDDI registry that acts as the store, but also how WS-Federation maps to the UDDI data model.

As section 3.2 describes, a given web service may advertise related attribute stores using the `RelatedService` policy assertion. To indicate that the attribute store can be accessed using the UDDI protocol, the `RelatedService` MUST indicate the version-specific UDDI portType name inside the endpoint reference. The following illustrates a `RelatedService` policy assertion that references a UDDI v2 endpoint:

```
<wsse:RelatedService Type="wsse:ServiceAS"
   xmlns:uddi="urn:uddi-org:inquiry_v2">
  <wsa:Address>http://www.fabrikam123.com/users</wsa:Address>
  <wsa:PortType>uddi:Inquire</wsa:PortType>
</wsse:RelatedService>
```

In terms of mapping the data model of a WS-Federation attribute store to the UDDI data model, each principal is expected to have a unique tModel that is queryable via UDDI and may have associated attributes that are also queryable. To distinguish tModels that represent WS-Federation principals from other entities in the registry, this specification defines a classification tModel (Principal Attribute Taxonomy tModel) that all principal tModels must carry in its categoryBag.

Principal Attribute taxonomy tModel:

UDDI v1 and v2 key:uuid:1a7d9432-22e9-3377-a609-fcde7930cab4

UDDI v3 key: uddi:schemas.xmlsoap.org:federatedprincipalattribute:2003_06

Each related set of attributes for a principal are saved in one tModel and are listed in the categoryBag of that tModel using references to the Principal Attribute taxonomy tModel. Each attribute, is then a keyedReference in the categoryBag of the tModel with the tModelKey below and the attribute name maps to the keyName attribute and the attribute value maps to the keyValue attribute

```
<uddi:keyedReference xmlns:uddi=”urn:uddi-org:api_v2”
    tModelKey=”uuid:1a7d9432-22e9-3377-a609-fcde7930cab4”
    keyName=”AttributeName”
    keyValue=”AttributeValue”/>
```

It is important to note that the access to the attributes on the tModel is determined by the policy of the UDDI registry where they are published. By default, most UDDI registries allow read access to all UDDI meta data to all users authorized to the registry. To restrict access to the tModel it is then necessary for the implementation of UDDI being used to have the capability to apply a user specified access control policy using mechanisms from WS-Policy attachment.

A policy aware registry could restrict access to the tModel by applying the policy attached to the tModel in the categoryBag of the tModel. WS-Policy Attachment defines the attachment of a given policy at a URL as follows (where uuid:0b1b5a47-bebf-3b7d-9802-f2dd80a91ade is the tModelKey for the WS-Policy Attachment category system):

```
<uddi:keyedReference xmlns:uddi=”urn:uddi-org:api_v2”
    keyName="Policy expression for access to this tModel"
    keyValue="http://www.example.com/mytmodel/policy"
    tModelKey="uuid:0b1b5a47-bebf-3b7d-9802-f2dd80a91ade" />
```

Using the relatedService reference, these attributes contained in the tModels are found using the UDDI businessService that is returned from the query to the RelatedService above. This businessService SHOULD contain one or more bindings to tModels which include the attributes for the principal. To obtain the attributes, subsequent get_tModel requests should be issued to the tModelKeys representing each the tModel for each attribute set for the principal.

6. Pseudonym Service

The pseudonym service is a special type of attribute service which maintains alternate identity information (and optionally associated tokens) for principals in the attribute service.

Pseudonym services may exist in some form already in service environments using service-specification protocols. This specification defines an additional, generic, interface to these services for interoperability with Web services.

The figure below illustrates the conceptual namespace of a pseudonym service:

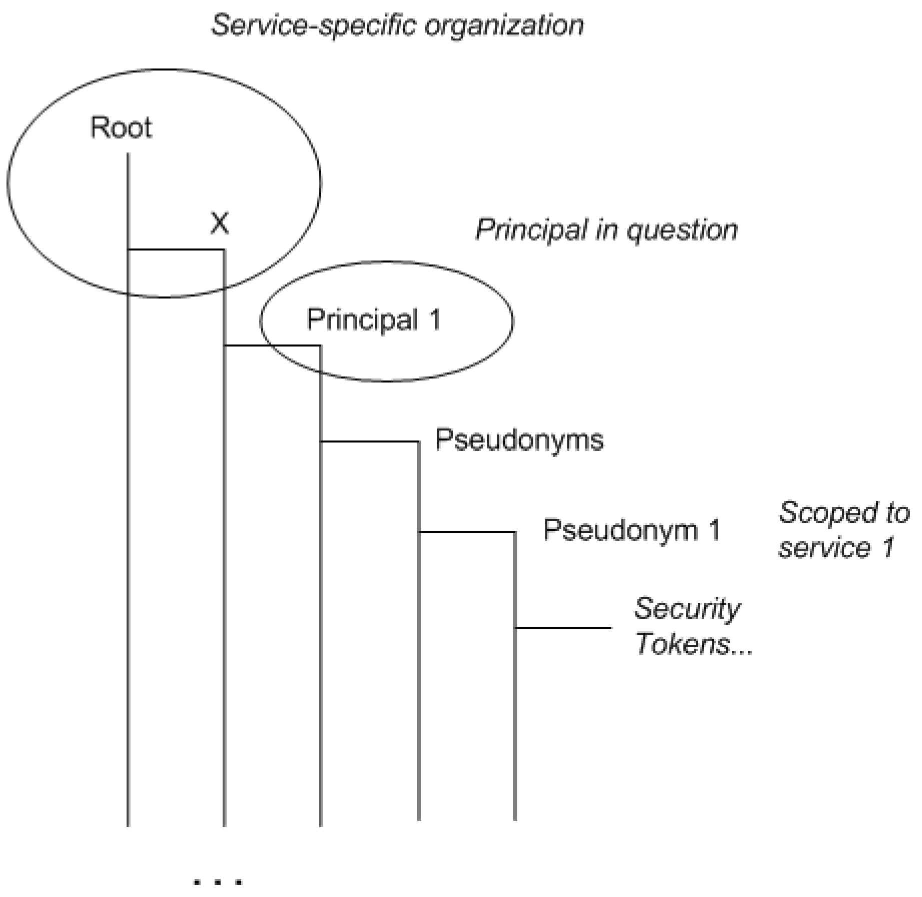

Figure 15 Pseudonym Service

The service MAY provide some level of organization or context. That is, this specification makes no proposals or requirements on the organization of the data, just that a principal exist and be addressable using the mechanisms described here.

Within the namespace principals are associated and a set of zero or more pseudonyms defined. Each pseudonym MAY be scoped, that is, each pseudonym may have a scope to which it applies (possibly more than one resource/service).

A pseudonym MAY have zero or more associated security tokens. This is an important aspect because it allows an IP to directly return the appropriate token for specified scopes. For example, when Fred.Jones requested a token for Fabrikam123.com, his IP could have returned the Freddo identity directly allowing the requestor to pass this to Fabrikam123. This approach is more efficient and allows for greater privacy options.

It is expected that different pseudonyms may have different access control and privacy policies for each subject that needs. Consequently, each pseudonym SHOULD be capable of expressing its own access control and privacy policy via WS-Policy. As well, the access control and privacy policy SHOULD take into account the associated scope(s) and principals that can speak for the scope(s).

Pseudonym services MUST support the following interfaces for getting, setting, and deleting pseudonyms:

6.1. Getting Pseudonyms

Pseudonyms are requested from a pseudonym service using the `<wsse:GetPseudonym>` request.

The syntax for the `<wsse:GetPseudonym>` element is as follows:

```
<wsse:GetPseudonym>
    <wsse:PseudonymBasis>...</wsse:PseudonymBasis>
    <wsse:RelativeTo>...</wsse:RelativeTo>
</wsse:GetPseudonym>
```

The following describes elements and attributes used in a `<wsse:GetPseudonym>` element.

/GetPseudonym
: This element indicates a request for a pseudonym based on given identity information.

/GetPseudonym/PseudonymBasis
: This element specifies a security token or security token reference identifying the known identity information.

/GetPseudonym/RelativeTo
: The required element indicates the scope for which the pseudonym is requested. This element has the same type as `<wsp:AppliesTo>`.

Pseudonyms are returned in the `<wsu:GetPseudonymResponse>` element as follows:

```
<wsse:GetPseudonymResponse>
   <wsee:RelativeTo>...</wsee:RelativeTo>
   <wsse:Pseudonym>
     <wsu:Expires>...</wsu:Expires>
     <wsse:SecurityToken>...</wsse:SecurityToken>
     <wsse:ProofToken>...</wsse:ProofToken>
     ...
   </wsse:Pseudonym>
</wsse:GetPseudonymResponse>
```

The following describes elements and attributes used in a `<wsse:GetPseudonymResponse>` element.

/GetPseudonymResponse
: The element indicates a returned pseudonym. Either this or a Fault is returned.

/GetPseudonymResponse/RelativeTo
: The required element indicates the scope to which the pseudonym applies. This element has the same type as `<wsp:AppliesTo>`.

/GetPseudonymResponse/Pseudonym
: This element represents a pseudonym for a principal.

/GetPseudonymResponse/Pseudonym/wse:Expires

This optional element indicates the expiration of the pseudonym.

/GetPseudonymResponse/Pseudonym/SecurityToken

This optional element indicates a security token for the scope. Note that multiple tokens MAY be specified.

/GetPseudonymResponse/Pseudonym/ProofToken

This optional element indicates a proof token for the scope. Note that multiple tokens MAY be specified.

/GetPseudonymResponse/Pseudonym/{any}

This is an extensibility mechanism to allow additional information to be specified within the pseudonym.

For example, the following example obtains the "Nick" pseudonym of the principal identified by http://www.fabrikam123.com/NNK.

```
<S:Envelope>
  ...
  <S:Body>
    <wsse:GetPseudonym>
        <wsse:PseudonymBasis>
            <wsse:BinarySecurityToken>...<wsse:BinarySecurityToken>
        <wsse:PseudonymBasis>
        <wsse:RelativeTo>
          <wsa:Address>
                http://www.fabrikam123.com/NNK</wsa:Address>
        </wsse:RelativeTo>
    </wsse:GetPseudonym>
  </S:Body>
</S:Envelope>
```

A sample response might be as follows:

```
<S:Envelope>
  ...
  <S:Body>
    <wsse:GetPseudonymResponse>
        <wsse:RelativeTo>
            <wsp:Address URI="http://www.busines456.com"/>
        </wsse:RelativeTo>
        <wsse:Pseudonym>
          <wsu:Expires>2003-12-10T09:00Z</wsu:Expires>
          <wsse:SecurityToken>...
             </wsse:SecurityToken>
          <wsse:ProofToken>...</wsse:ProofToken>
        </wsse:Pseudonym>
    </wsse:GetPseudonymResponse>
  </S:Body>
</S:Envelope>
```

6.2. Setting Pseudonyms

Pseudonyms are updated in a pseudonym service using the `<wsse:SetPseudonym>` request.

The syntax for the `<wsse:SetPseudonym>` element is as follows:

```
<wsse:SetPseudonym>
    <wsse:PseudonymBasis>...</wsse:PseudonymBasis>
    <wsse:RelativeTo>...</wsse:RelativeTo>
    <wsse:Pseudonym>...</wsse:Pseudonym>
</wsse:SetPseudonym>
```

The following describes elements and attributes used in a `<wsse:SetPseudonym>` element.

/SetPseudonym
: This element indicates a pseudonym update request.

/SetPseudonym/PseudonymBasis
: This element specifies a security token or security token reference indicating the token used as the basis for the pseudonym mapping.

/SetPseudonym/RelativeTo
: The required element indicates the scope to which the pseudonym applies. This element has the same type as `<wsp:AppliesTo>`.

/SetPseudonym/Pseudonym
: The required element specified the value of the pseudonym. If a pseudonym is present for the scope, it is replaced.

The result is returned in a `<wsu:SetPseudonymResponse>` element as follows:

```
<wsse:SetPseudonymResponse/>
```

The following describes elements and attributes used in a `<wsse:SetPseudonymResponse>` element.

/SetPseudonymResponse
: The element indicates a successful response to a pseudonym update request. Either this or a Fault is returned.

The following example sets the "Nick" pseudonym of the principal identified by http://www.fabrikam123.com/NNK.

```
<S:Envelope>
```

```
  ...
  <S:Body>
    <wsse:SetPseudonym>
        <wsse:PseudonymBasis>
            <wsse:BinarySecurityToken>...<wsse:BinarySecurityToken>
        <wsse:PseudonymBasis>
        <wsse:RelativeTo>
          <wsa:Address>
                http://www.fabrikam123.com/NNK</wsa:Address>
        </wsse:RelativeTo>
        <wsse:Pseudonym>
          <wsse:SecurityToken>
              <wsse:UsernameToken>
                  <Username> "Nick" </Userename>
              </wsse:UsernameToken>
          </wsse:SecurityToken>
          <wsse:ProofToken>...</wsse:ProofToken>
        </wsse:Pseudonym>
    </wsse:SetPseudonym>
  </S:Body>
</S:Envelope>
```

A sample response might be as follows:

```
<S:Envelope>
  ...
  <S:Body>
    <wsse:SetPseudonymResponse/>
  </S:Body>
</S:Envelope>
```

6.3. Deleting Pseudonyms

Pseudonyms are deleted in a pseudonym service using the `<wsse:DeletePseudonym>` request.

The syntax for the `<wsse:DeletePseudonym>` element is as follows:

```
<wsse:DeletePseudonym>
    <wsse:PseudonymBasis>...</wsse:PseudonymBasis>
    <wsse:RelativeTo>...</wsse:RelativeTo>
    <wsse:Pseudonym>...</wsse:Pseudonym>
</wsse:DeletePseudonym>
```

The following describes elements and attributes used in a `<wsse:DeletePseudonym>` element.

/DeletePseudonym
: This element indicates a pseudonym update request.

/DeletePseudonym/PseudonymBasis
: This element specifies a security token or security token reference indicating the token used as the basis for the pseudonym mapping.

/DeletePseudonym/RelativeTo
: This required element indicates the scope to which the pseudonym applies. This element has the same type as `<wsp:AppliesTo>`.

/DeletePseudonym/Pseudonym
: This optional element specifies the value of the pseudonym. If this isn't specified, then all pseudonyms for the scope are removed – effectively a request to defederate.

The result is returned in a `<wsu:DeletePseudonymResponse>` element as follows:

```
<wsse:DeletePseudonymResponse/>
```

The following describes elements and attributes used in a `<wsse:DeletePseudonymResponse>` element.

/DeletePseudonymResponse
: The element indicates a successful response to a pseudonym delete request. Either this or a Fault is returned.

The following example deletes the "Nick" pseudonym of the principal identified by http://www.fabrikam123.com/NNK.

```
<S:Envelope>
  ...
  <S:Body>
    <wsse:DeletePseudonym>
        <wsse:PseudonymBasis>
            <wsse:BinarySecurityToken>...<wsse:BinarySecurityToken>
        <wsse:PseudonymBasis>
        <wsse:RelativeTo>
          <wsa:Address>
                http://www.fabrikam123.com/NNK</wsa:Address>
        </wsse:RelativeTo>
        <wsse:Pseudonym>
```

```
          <wsse:SecurityToken>
              <wsse:UsernameToken>
                  <Username> "Nick" </Userename>
              </wsse:UsernameToken>
          </wsse:SecurityToken>
          <wsse:ProofToken>...</wsse:ProofToken>
        </wsse:Pseudonym>
    </wsse:DeletePseudonym>
  </S:Body>
</S:Envelope>
```

A sample response might be as follows:

```
<S:Envelope>
  ...
  <S:Body>
    <wsse:DeletePseudonymResponse/>
  </S:Body>
</S:Envelope>
```

7. Security Tokens and Pseudonyms

As previously mentioned, the pseudonym service can also be used to store tokens associated with the pseudonym. Cooperating Identity Providers and security token services can then be used to automatically obtain the pseudonyms and tokens based on security token requests for scopes associated with the pseudonyms.

In figure 16 below illustrates two examples of how security tokens are associated with resources/services. In the figure on the left, the requestor first obtains the security token(s) from the IP/STS for the resource/service and then saves them in the pseudonym service. The pseudonyms can be obtained from the pseudonym service prior to subsequent communication with the resource removing the need for the resource's IP/STS to communicate with the requestor's pseudonym service. The figure on the right illustrates the scenario where IP/STS for the resource/service associates the security token(s) for the requestor as needed and looks them up (as illustrated in previous sections).

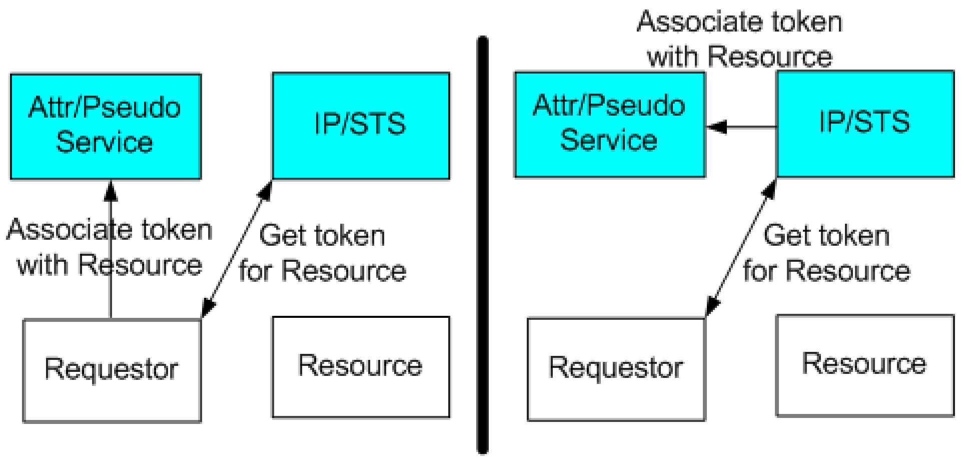

Figure 16 Attribute & Pseudonym Services

However when the requestor requests tokens for a resource/service, using a WS-Trust `<RequestSecurityToken>` whose scope has an associated pseudonym/token, it is returned as illustrated below in the `<RequestSecurityTokenResponse>` which can then be used when communicating with the resource:

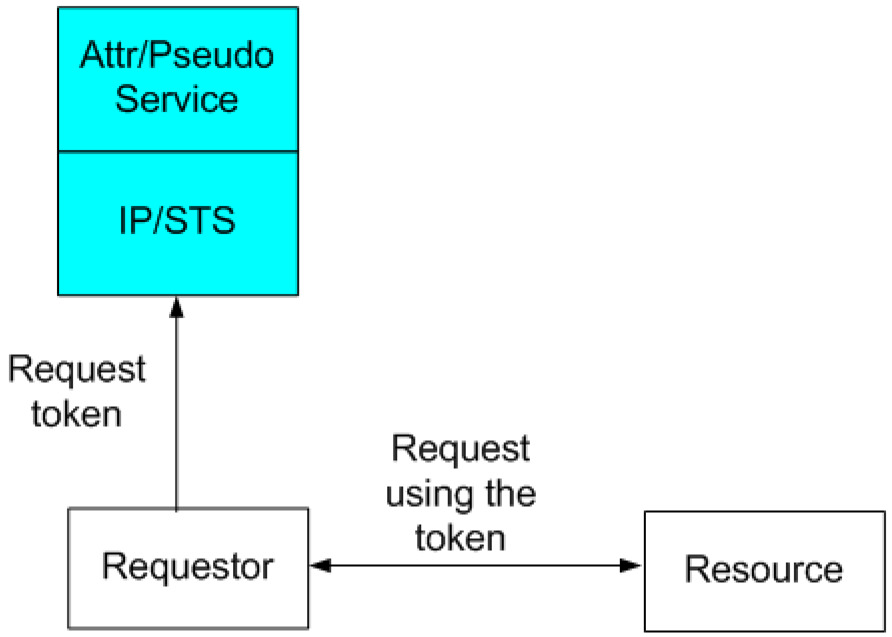

Figure 17 Attribute Service

The pseudonym service SHOULD be self-maintained with respect to valid security tokens. That is, security tokens that have expired or are otherwise not valid for any reason MAY be automatically be discarded by the service.

This approach is an alternative to having the pseudonym service directly return the security token issuance. Both approaches need to be available in order to address different scenarios and requirements.

The following sub-sections describe how token issuance works for different types of keys.

7.1. RST and RSTR Extensions

With the addition of pseudonyms and the integration of an IP/STS with a pseudonym service, there are additional options that MAY be included in the security token requests using the `<wsse:RequestSecurityToken>` request. The following syntax illustrates the RST extension to support these new options:

```
<RequestPseudonym SingleUse="..." Lookup="..."/>
```

/RequestPseudonym
: This optional element MAY be specified in a `<wsse:RequestSecurityToken>` request to indicate how pseudonyms are to be processed for the requested token.

/RequestPseudonym/@SingleUse
: This optional attribute indicates if a single-use pseudonym is returned (true), or if the service uses a constant identifier (false – the default).

/RequestPseudonym/@Lookup
: This optional attribute indicates if an associated pseudonym for the specified scope is used (true – the default) or if the primary identity is used even if an appropriate pseudonym is associated (false).

/Pseudonym/{any}
: This is an extensibility mechanism to allow additional information to be specified.

/Pseudonym/@{any}
: This is an extensibility mechanism to allow additional attributes to be specified.

If the `<RequestPseudonym>` isn't present, pseudonym usage and single use is at the discretion of the IP/STS.

7.2. Usernames and Passwords

If a IP/STS returns a security token based on a username, then the token can be stored in the pseudonym service.

If a corresponding password is issued (or if the requestor specified one), then it too MAY be stored with the pseudonym and security token so that it can be returned as the proof-of-possession token in the RSTR response.

If a pseudonym is present, but no security token is specified, then the IP/STS MAY return a `<UsernameToken>` in the RSTR response to indicate the pseudonym.

7.3. Public Keys

Generally, when a IP/STS issues a new security token with public key credentials, the public key in the new security token is the same as the key in the provided input security token thereby allowing the same proof (private key) to be used with the new token since the public key is the same. In such cases, the new token can be saved directly.

If, however, the IP/STS issues a new public key (and corresponding private key), then the private key MAY be stored with the pseudonym as a proof token so that it can be returned as the proof-of-possession token in the RSTR response.

7.4. Symmetric Keys

If an IP/STS returns a token based on a symmetric key (and the corresponding proof information), then the proof information MAY be stored with the pseudonym and token so that it can be used to construct a proof-of-possession token in the RSTR response.

8. Error Handling

This specification defines the following error codes that MAY be used. Other errors MAY also be used.

Error that occurred	**faultcode**
No pseudonym found for the specified scope	`wsse:NoPseudonymInScope`

9. Security Considerations

It is strongly RECOMMENDED that the communication between services be secured using the mechanisms described in WS-Security. In order to properly secure messages, the body and all relevant headers need to be included in the signature.

All federation-related messages such as sign-out, principal, attribute, and pseudonym management SHOULD be signed to ensure integrity. If a message is received where the body is not signed, it is RECOMMENDED that the message not be processed.

All sign-out requests MUST be signed by the principal being purported to be signing in or out, or by a principal that is authorized to be on behalf of the indicated principal.

It is also RECOMMENDED that all messages be signed by the appropriate security token service. If a message is received that does not have a signature from a principal authorized to speak for the security token service, it is RECOMMENDED that the message not be processed.

The attribute service maintains information that may be very sensitive. Significant care should be taken to ensure that a principal's privacy is taken into account first and foremost.

The pseudonym service MAY contain passwords or other information used in proof-of-possession mechanisms. Extreme care needs to be taken with this data to ensure that it cannot be compromised. It is strongly RECOMMENDED that such information be encrypted over communications channels and in any physical storage.

If a security token does not contain an embedded signature (or similar integrity mechanism), it SHOULD be included in any message integrity mechanisms.

If privacy is a concern, the security tokens MAY be encrypted for the authorized recipient(s) using mechanisms in WS-Security.

The following list summarizes common classes of attacks that apply to this protocol and identifies the mechanism to prevent/mitigate the attacks:

- **Message alteration** – Alteration is prevented by including signatures of the message information using WS-Security.
- **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using WS-Security.
- **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by comparing secured policies – see WS-Policy and WS-SecurityPolicy).

- **Authentication** – Authentication is established using the mechanisms described in WS-Security and WS-Trust. Each message is authenticated using the mechanisms described in WS-Security.
- **Accountability** – Accountability is a function of the type of and string of the key and algorithms being used. In many cases, a strong symmetric key provides sufficient accountability. However, in some environments, strong PKI signatures are required.
- **Availability** – All reliable messaging services are subject to a variety of availability attacks. Replay detection is a common attack and it is RCOMMENDED that this be addressed by the mechanisms described in WS-Security. Other attacks, such as network-level denial of service attacks are harder to avoid and are outside the scope of this specification. That said, care should be taken to ensure that minimal state is saved prior to any authenticating sequences.
- **Replay attacks:** It is possible that requests for security tokens could be replayed. Consequently, it is RECOMMENDED that all communication between security token services and resources take place over secure connections. All cookies indicating state SHOULD be set as secure.
- **Forged security tokens:** Security token services MUST guard their signature keys to prevent forging of tokens and requestor identities.
- **Privacy:** Security token services SHOULD NOT send requestors' personal identifying information or information without getting consent from the requestor. For example a Web site SHOULD NOT receive requestors' personal information without an appropriate consent process.
- **Compromised services:** If a security token service is compromised, all requestor accounts serviced SHOULD be assumed to be compromised as well (since an attacker can issue security tokens for any account they want). However they MUST NOT be able to issue tokens directly for identities outside the compromised realm. This is of special concern in scenarios like the 3rd-party brokered trust where a 3rd party IP/STS is brokering trust between two realms. In such a case compromising the broker results in the ability to indirectly issue tokens for another realm by indicating trust.

As with all communications careful analysis should be performed on the messages and interactions to ensure they meet the desired security requirements.

10. Notes

[1] WS-MetadataExchange is a set of Web service mechanisms to exchange policies, WSDL, schema and other metadata between two or more parties. This specification is part of the Web services roadmap for both WS-ReliableMessaging and WS-Federation. WS-MetadataExchange will be published this summer.

[2] A supplemental profile document or revision to this document, to be published this summer, will address interoperability concerns between attribute services.

11. Acknowledgements

This specification has been developed as a result of joint work with many individuals and teams, including:

Josh Gray, Microsoft
Tim Hahn, IBM

Andrew Hatley, IBM
Heather Hinton, IBM
Bronislav Kavsan, RSA Security
Brad Lovering, Microsoft
Anthony Moran, IBM
Birgit Pfitzmann, IBM
Robert Philpott, RSA Security
Yordan Rouskov, Microsoft
Jeff Spelman, Microsoft
Shane Weeden, IBM

12. References

[KEYWORDS]
S. Bradner, "Key Words for Use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997.

[HTTP]
R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, RFC 2616, "Hypertext Transfer Protocol -- HTTP/1.1". June 1999.

[SOAP]
W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.
Draft, SOAP 1.2, http://www.w3.org/TR/soap12-part0/
Draft, SOAP 1.2, http://www.w3.org/TR/soap12-part1/
Draft, SOAP 1.2, http://www.w3.org/TR/soap12-part2/

[URI]
T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[WS-Security]
"Web Services Security Language", IBM, Microsoft, VeriSign, April 2002.
"WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002.
"WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002

[WS-Policy]
"Web Services Policy Framework", BEA, IBM, Microsoft, SAP, December 2002

[WS-PolicyAttachment]
"Web Services Policy Attachment Language", BEA, IBM, and Microsoft, SAP, December 2002

[WS-PolicyAssertions]
"Web Services Policy Assertions Language", BEA, IBM, Microsoft, SAP, December 2002

[WS-Trust]
"Web Services Trust Language", IBM, Microsoft, RSA, VeriSign, December 2002

[WS-SecureConversation]
"Web Services Secure Conversation Language", IBM, Microsoft, RSA, VeriSign, December 2002

[WS-SecurityPolicy]
"Web Services Security Policy Language", IBM, Microsoft, RSA, Verisign December 2002

[WS-FederationActive]
"Web Services Federation Language: Active Requestor Profile", BEA, IBM, Microsoft, RSA Security, VeriSign, July 2003

[WS-FederationPassive]
"Web Services Federation Language: Passive Requestor Profile", BEA, IBM, Microsoft, RSA Security, VeriSign, July 2003

[WS-ReliableMessaging]
"Web Services Reliable Messaging Protocol", BEA, IBM, Microsoft, Tibco, February 2003

[XML-ns]
W3C Recommendation, "Namespaces in XML," 14 January 1999.

Appendix I - WSDL

The following illustrates the WSDL for the Web service methods described in this specification:

```
<wsdl:definitions
   targetNamespace="http://schemas.xmlsoap.org/ws/2003/07/secext"
    xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/07/secext"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
<!-- this is the WS-I BP-compliant way to import a schema -->
      <wsdl:types>
          <xs:schema>
              <xs:import
              namespace="http://schemas.xmlsoap.org/ws/2003/07/secext"
              schemaLocation="secext.xsd"/>
          </xs:schema>
      </wsdl:types>



<!-- These are the base messages -->

      <wsdl:message name="SignOutMsg">
            <wsdl:part name="signout" element="wsse:SignOut" />
      </wsdl:message>
      <wsdl:message name="RequestSSOMessagesMsg">
            <wsdl:part name="requestsso"
                             element="wsse:RequestSSOMessages" />
      </wsdl:message>
      <wsdl:message name="CancelSSOMessagesMsg">
            <wsdl:part name="cancelsso"
                              element="wsse:CancelSSOMessages" />
      </wsdl:message>
      <wsdl:message name="GetPseudonymMsg">
            <wsdl:part name="getpseudorequest"
                              element="wsse:GetPseudonym" />
      </wsdl:message>
      <wsdl:message name="GetPseudonymResponseMsg">
            <wsdl:part name="getpseudoresponse"
                              element="wsse:GetPseudonymResponse" />
      </wsdl:message>
      <wsdl:message name="SetPseudonymMsg">
            <wsdl:part name="setpseudorequest"
                              element="wsse:SetPseudonym" />
      </wsdl:message>
      <wsdl:message name="SetPseudonymResponseMsg">
            <wsdl:part name="Setpseudoresponse"
                              element="wsse:SetPseudonymResponse" />
      </wsdl:message>
      <wsdl:message name="DeletePseudonymMsg">
            <wsdl:part name="delpseudorequest"
                              element="wsse:DeletePseudonym" />
      </wsdl:message>
      <wsdl:message name="DeletePseudonymResponseMsg">
            <wsdl:part name="delpseudoresponse"
                              element="wsse:DeletePseudonymResponse" />
      </wsdl:message>



<!-- These portTypes model the Signout messages -->

      <wsdl:portType name="SignOutService">
              <wsdl:operation name="SignOut">
                    <wsdl:input message="wsse:SignOutMsg"/>
              </wsdl:operation>
        </wsdl:portType>
      <wsdl:portType name="RequestSSOService">
              <wsdl:operation name="RequestSSO">
                    <wsdl:input
                                message="wsse:RequestSSOMessagesMsg"/>
              </wsdl:operation>
        </wsdl:portType>

      <wsdl:portType name="CancelSSOService">
              <wsdl:operation name="CancelSSO">
                    <wsdl:input
                                  message="wsse:CancelSSOMessagesMsg"/>
              </wsdl:operation>
        </wsdl:portType>



<!-- These portTypes model the Pseudonym messages -->

    <wsdl:portType name="PseudonymRequester">
              <wsdl:operation name="GetPseudonymResponse">
                    <wsdl:input
                               message="wsse:GetPseudonymResponseMsg"/>
              </wsdl:operation>
              <wsdl:operation name="SetPseudonymResponse">
                    <wsdl:input
                              message="wsse:SetPseudonymResponseMsg"/>
              </wsdl:operation>
              <wsdl:operation name="DeletePseudonymResponse">
                    <wsdl:input
                           message="wsse:DeletePseudonymResponseMsg"/>
              </wsdl:operation>
    </wsdl:portType>

      <wsdl:portType name="PseudonymRequestService">
              <wsdl:operation name="GetPseudonymRequest">
                    <wsdl:input message="wsse:GetPseudonymMsg"/>
              </wsdl:operation>
              <wsdl:operation name="SetPseudonymRequest">
                    <wsdl:input message="wsse:SetPseudonymMsg"/>
              </wsdl:operation>
              <wsdl:operation name="DeletePseudonymRequest">
                    <wsdl:input
                                  message="wsse:DeletePseudonymMsg"/>
              </wsdl:operation>
    </wsdl:portType>

    <wsdl:portType name="PseudonymService">
              <wsdl:operation name="GetPsuedonym">
                    <wsdl:input message="wsse:GetPseudonymMsg"/>
                    <wsdl:output
                               message="wsse:GetPseudonymResponseMsg"/>
              </wsdl:operation>
              <wsdl:operation name="SetPsuedonym">
                    <wsdl:input message="wsse:SetPseudonymMsg"/>
                    <wsdl:output
                              message="wsse:SetPseudonymResponseMsg"/>
              </wsdl:operation>
              <wsdl:operation name="DeletePsuedonym">
                    <wsdl:input
                                message="wsse:DeletePseudonymMsg"/>
                    <wsdl:output
                            message="wsse:DeletePseudonymResponseMsg"/>
              </wsdl:operation>
    </wsdl:portType>

</wsdl:definitions>
```