Web Services Federation Language (WS-Federation)

Version 1.1
December 2006

Authors

Hal Lockhart, BEA
Steve Andersen, BMC Software
Jeff Bohren, BMC Software
Yakov Sverdlov, CA Inc.
Maryann Hondo, IBM
Hiroshi Maruyama, IBM
Anthony Nadalin (Editor), IBM
Nataraj Nagaratnam, IBM
Toufic Boubez, Layer 7 Technologies, Inc.
K Scott Morrison, Layer 7 Technologies, Inc.
Chris Kaler (Editor), Microsoft
Arun Nanda, Microsoft
Don Schmidt, Microsoft
Doug Walters, Microsoft
Hervey Wilson, Microsoft
Lloyd Burch, Novell, Inc.
Doug Earl, Novell, Inc.
Siddharth Baja, VeriSign
Hemma Prafullchandra, VeriSign

Copyright Notice

Abstract

This specification defines mechanisms to allow different security realms to federate, such that authorized access to resources managed in one realm can be provided to security principals whose identities and attributes are managed in other realms. This includes mechanisms for brokering of identity, attribute, authentication and authorization assertions between realms, and privacy of federated claims.

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, transport, and application-specific protocols to accommodate a wide variety of security models.

Status

This WS-Federation Specification is a public draft release and is provided for review and evaluation only. The Authors hope to solicit your contributions and suggestions in the near future. The Authors 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, such that authorized access to resources managed in one realm can be provided to security principals whose identities are managed in other realms. While the final access control decision is enforced strictly by the realm that controls the resource, federation provides mechanisms that enable the decision to be based on the declaration (or brokering) of identity, attribute, authentication and authorization assertions between realms. The choice of mechanisms, in turn, is dependent upon trust relationships between the realms. While trust establishment is outside the scope of this document, the use of metadata to help automate the process is discussed.

A general federation framework must be capable of integrating existing infrastructures into the federation without requiring major new infrastructure investments. This means that the types of security tokens and infrastructures can vary as can the attribute stores and discovery mechanisms. Additionally, the trust topologies, relationships, and mechanisms can also vary requiring the federation framework to support the resource's approach to trust rather than forcing the resource to change.

The federation framework defined in this specification builds on WS-Security, WS-Trust, and the WS-* family of specifications providing a rich extensible mechanism for federation. The WS-Security and WS-Trust specification allow for different types of security tokens, infrastructures, and trust topologies. This specification uses these building blocks to define additional federation mechanisms that extend these specifications and leverage other WS-* specifications.

The mechanisms defined in this specification can be used by Web service (SOAP) requestors as well as Web browser requestors. The Web service requestors are assumed to understand the WS-Security and WS-Trust mechanisms and be capable of interacting directly with Web service providers. The Web browser mechanisms describe how the WS-* messages (e.g. WS-Trust's RST and RSTR) are encoded in HTTP messages such that they can be passed between resources and Identity Provider (IP)/ Security Token Service (STS) parties by way of a Web browser client. This definition allows the full richness of WS-Trust, WS-Policy, and other WS-* mechanisms to be leveraged in Web browser environments.

It is expected that WS-Policy and WS-SecurityPolicy (as well as extensions in this specification) are used to describe what aspects of the federation framework are required/supported by federation participants and that this information is used to determine the appropriate communication options.

1.1. Document Roadmap

The remainder of this section describes the goals, conventions, namespaces, schema and WSDL locations, and terminology for this document.

Chapter 2 provides an overview of the federation model. This includes a discussion of the federation goals and issues, different trust topologies, identity mapping, and the components of the federation framework.

Chapter 3 describes the overall federation metadata model and how it is used within the federation framework. This includes how it is expressed and obtained within and across federations.

Chapter 4 describes the optional sign-out mechanisms of the federation framework. This includes how sign-out messages are managed within and across federations including the details of sign-out messages.

Chapter 5 describes the role of attribute services in the federation framework.

Chapter 6 defines the pseudonym service within the federation framework. This includes how pseudonyms are obtained, mapped, and managed.

Chapter 7 presents how pseudonyms can be directly integrated into security token services by extending the token request and response messages defined in WS-Trust.

Chapter 8 introduces additional extensions to WS-Trust that are designed to facilitate federation and includes the use of token references, federation selection, extraction of keys for different trust styles, and different authentication types.

Chapter 9 describes federated authorization including extensions to WS-Trust and minimum requirements.

Chapter 10 describes how specific policy and metadata can be provided for a specific message pattern and during normal requestor/recipient interactions.

Chapter 11 describes pre-defined types of authentication for use with WS-Trust.

Chapter 12 describes extensions to WS-Trust for privacy of security token claims and how privacy statements can be made in federated metadata documents.

Chapter 13 describes how WS-Federation and WS-Trust can be used by web browser requestors and web applications that do not support direct SOAP messaging.

Chapter 14 describes extensions to WS-SecurityPolicy to allow federation participants to indicate additional federation requirements.

Chapters 15 and 16 define federation-specific error codes and outline security considerations for architects, implementers, and administrators of federated systems.

Chapters 17 and 18 acknowledge contributors to the specification and all references made by this specification to other documents.

Appendix I provides a sample WSDL definition of the services defined in this specifications.

Appendix II provides a detailed example of the messages for a Web browser-based requestor that is using the federation mechanisms described in chapter 9.

Appendix III describes several additional use cases motivating the federation framework for both SOAP-based and Web browser-based requestors.

1.2. Goals and Requirements

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

1.2.1 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
- Allow federation using different types of security tokens, trust topologies, and security infrastructures
- Facilitate brokering of trust and security token exchange for both SOAP requestors and Web browsers using common underlying mechanisms and semantics
- Express federation metadata to facilitate communication and interoperability between federation participants
- Allow identity mapping to occur at either requestor, target service, or any IP/STS

- Provide identity mapping support if target services choose to maintain optional local identities, but do not require local identities
- Allow for different levels of privacy for identity (e.g. different forms and uniqueness of digital identities) information and attributes
- Allow for authenticated but anonymous federation

1.2.2 Non-Goals

The following topics are outside the scope of this document:

- Definition of message security (see WS-Security)
- Trust establishment/verification protocols (see WS-Trust)
- Management of trust or trust relationships
- Specification of new security token formats beyond token references
- Specification of new attribute store interfaces beyond UDDI
- Definition of new security token assertion/claim formats
- Requirement on specific security token formats
- Requirement on specific types of trust relationships
- Requirement on specific types of account linkages
- Requirement on specific types of identity mapping

1.3. 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 [KEYWORDS].

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/>`).

1.4. Namespaces

The following namespaces are used in this document:

Prefix	Namespace
fed	http://schemas.xmlsoap.org/ws/2006/12/federation
auth	http://schemas.xmlsoap.org/ws/2006/12/authorization
priv	http://schemas.xmlsoap.org/ws/2006/12/privacy
mex	http://schemas.xmlsoap.org/ws/2004/09/mex

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy
wsa	http://www.w3.org/2005/08/addressing
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsse11	http://docs.oasis-open.org/wss/oasis-wss-wsecurity-secext-1.1.xsd
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512
sp	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512
wsrt	http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer
wsxf	http://schemas.xmlsoap.org/ws/2004/09/transfer
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
ds	http://www.w3.org/2000/09/xmldsig#
xs	http://www.w3.org/2001/XMLSchema

It should be noted that the versions identified in the above table supersede versions identified in referenced specifications.

1.5. Schema and WSDL Files

The schemas for this specification can be located at:

http://schemas.xmlsoap.org/ws/2006/12/federation/federation.xsd

http://schemas.xmlsoap.org/ws/2006/12/authorization/authorization.xsd

http://schemas.xmlsoap.org/ws/2006/12/privacy/privacy.xsd

The WSDL for this specification can be located at:

http://schemas.xmlsoap.org/ws/2006/12/federation/federation.wsdl

1.6. Terminology

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

Association – The relationship established to uniquely link a principal across trust realms, despite the principal's having different identifiers in each trust realm. This is also referred to as "linked accounts" for the more narrowly scoped definition of associations (or linking).

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.

Authorization Service – A specialized type of Security Token Service (STS) that makes authorization decisions.

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

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

Digital Identity – A digital representation of a principal (or group of principals) that is unique to that principal (or group), and that acts as a reference to that principal (or group). For example, an email address MAY be treated as a digital identity, just as a machine's unique IP address MAY also be treated as a digital identity, or even a generated unique identifier. In the context of this document, the term *identity* is often used to refer to a *digital identity.* A principal may have multiple digital identities,

Digital Signature - A *digital signature* (of data or a message) is a value computed on the data/message (typically a hash) and protected with a cryptographic function. This has the effect of binding the digital signature to the data/message in such a way that intended recipients of the data can use the signature to verify that the data/message has not been altered since it was signed by the signer.

Digital Signature Validation – *Digital signature validation* is the process of verifying that digitally signed data/message has not been altered since it was signed.

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

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.

Federated Context – A group of realms to which a principal has established associations and to which a principal has presented Security Tokens and obtained session credentials. A federated context is dynamic, in that a realm is not part of the federated context if the principal has not presented Security Tokens. A federated context is not persistent, in that it does not exist beyond the principals (Single) Sign-Out actions.

Federation – A *federation* is a collection of realms that have established a producer-consumer relationship whereby one realm can provide authorized access to a resource it manages based on an identity, and possibly associated attributes, that are asserted in another realm. Federation requires trust such that a Relying Party can make a well-informed access control decision based on the credibility of identity and attribute data that is vouched for by another realm.

Federate – The process of establishing a federation between realms (partners). Associations are how principals create linkages between federated realms.

Identity Mapping – *Identity Mapping* is a method of creating relationships between digital identities or attributes associated with an individual principal by different Identity or Service Providers

Identity Provider (IP) – An *Identity Provider* is an entity that acts as an authentication service to end requestors and a data origin authentication service to service providers (this is typically an extension of a Security Token Service). Identity Providers (IP) are trusted (logical) 3rd parties which 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 service provider which may grant access to valuable resources and information based upon the integrity of the identity information provided by the IP.

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.

IP/STS – The acronym *IP/STS* is used to indicate a service that is either an Identity Provider (IP) or Security Token Service (STS).

Metadata – Any data that describes characteristics of a subject. For example, federation metadata describes attributes used in the federation process such as those used to identify – and either locate or determine the relationship to – a particular Identity Provider, Security Token Service or Relying Party service.

Metadata Endpoint Reference (MEPR) – A location expressed as an endpoint reference that enables a requestor to obtain all the required metadata for secure communications with a target service. This location may contain the metadata or a pointer to where it can be obtained.

Principal – An end user, an application, a machine, or any other type of entity that may act as a requestor. A principal is typically represented with a digital identity and may have multiple valid digital identities

***PII* – Personally identifying information** is any type of information that can be used to distinguish a specific individual or party, such as your name, address, phone number, or e-mail address**.**

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.

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.

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

Relying Party – A Web application or service that consumes Security Tokens issued by a Security Token Service.

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

Security Token Service (STS) - A *Security Token Service* is a Web service that provides issuance and management of security tokens (see [WS-Security] for a description of security tokens). That is, it makes security statements or claims often, although not required to be, in cryptographically protected sets. These statements are based on the

receipt of evidence that it can directly verify, or security tokens from authorities that it trusts. To assert trust, a service might prove its right to assert a set of claims by providing a security token or set of security tokens issued by an STS, or it could issue 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.

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.

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)

Sign-Out –The process by which a principal indicates that they will no longer be using their token and services in the realm in response to which the realm typically destroys their token caches and clear saved session credentials for the principal.

Single Sign-Out (SSO) – The process of sign-out in a federated context which involves notification to Security Token Services and Relying Parties to clear saved session credentials and Security Tokens.

SOAP Recipient – A *SOAP recipient* is an application that is capable of receiving Web services messages such as those described in WS-Security, WS-Trust, and this specification.

SOAP Requestor – A *SOAP requestor* is an application (possibly a Web browser) that is capable of issuing Web services messages such as those described in WS-Security, WS-Trust, and this specification.

Subset – A *subset* is a set of restrictions to limit options for interoperability.

Trust - *Trust is* the characteristic whereby one entity is willing to rely upon a second entity to execute a set of actions and/or to make a set of assertions about a set of principals and/or digital identities. In the general sense, trust derives from some relationship (typically a business or organizational relationship) between the entities. With respect to the assertions made by one entity to another, trust is commonly asserted by binding messages containing those assertions to a specific entity through the use of digital signatures and/or encryption.

Trust Realm/Domain - A *Trust Realm/Domain* 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 Domain/Realm.

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

Web Browser Requestor – A Web browser *requestor* is an HTTP browser capable of broadly supported [HTTP]. If a Web browser is not able to construct a SOAP message then it is often referred to as a *passive* requestor.

1.7. Compliance

An implementation is not compliant with this specification if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined herein. A SOAP Node MUST NOT use the XML namespace identifier for this specification (listed in Section 1.4) within SOAP Envelopes unless it is compliant with this specification.

This specification references a number of other specifications (see the table above). In order to comply with this specification, an implementation MUST implement the portions of referenced specifications necessary to comply with the required provisions of this specification. Additionally, the implementation of the portions of the referenced specifications that are specifically cited in this specification MUST comply with the rules for those portions as established in the referenced specification.

Additionally normative text within this specification takes precedence over normative outlines (as described in section 1.3), which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further constrains the schemas and/or WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for any other unrecognized/unsupported message.

2. Model

This chapter describes the overall model for federation building on the foundations specified in [WS-Security], [WS-SecurityPolicy], and [WS-Trust].

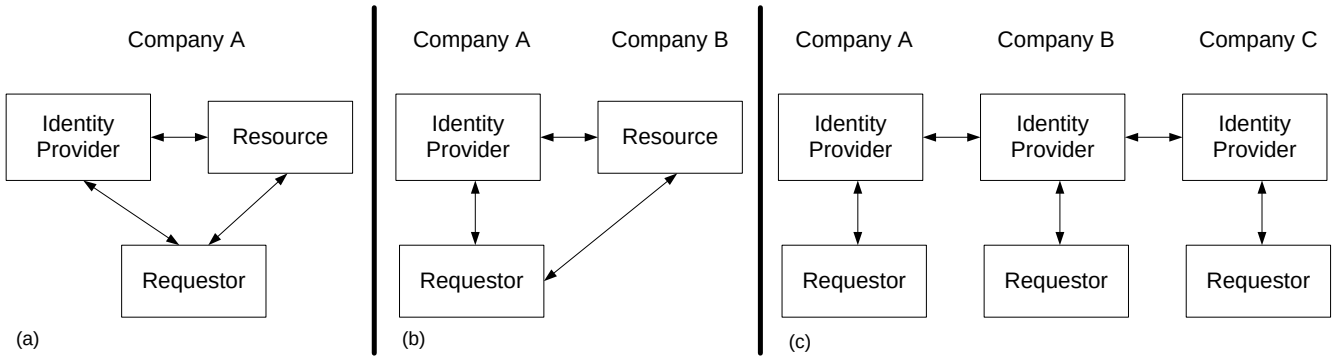
2.1. Federation Basics

The goal of federation is to allow security principal identities and attributes to be shared across trust boundaries according to established policies. The policies dictate, among other things, formats and options, as well as trusts and privacy/sharing requirements.

In the context of web services the goal is to allow these identities and attributes to be brokered from identity and security token issuers to services and other relying parties without requiring user intervention (unless specified by the underlying policies). This process involves the sharing of federation metadata which describes information about federated services, policies describing common communication requirements, and brokering of trust and tokens via security token exchange (issuances, validation, etc.).

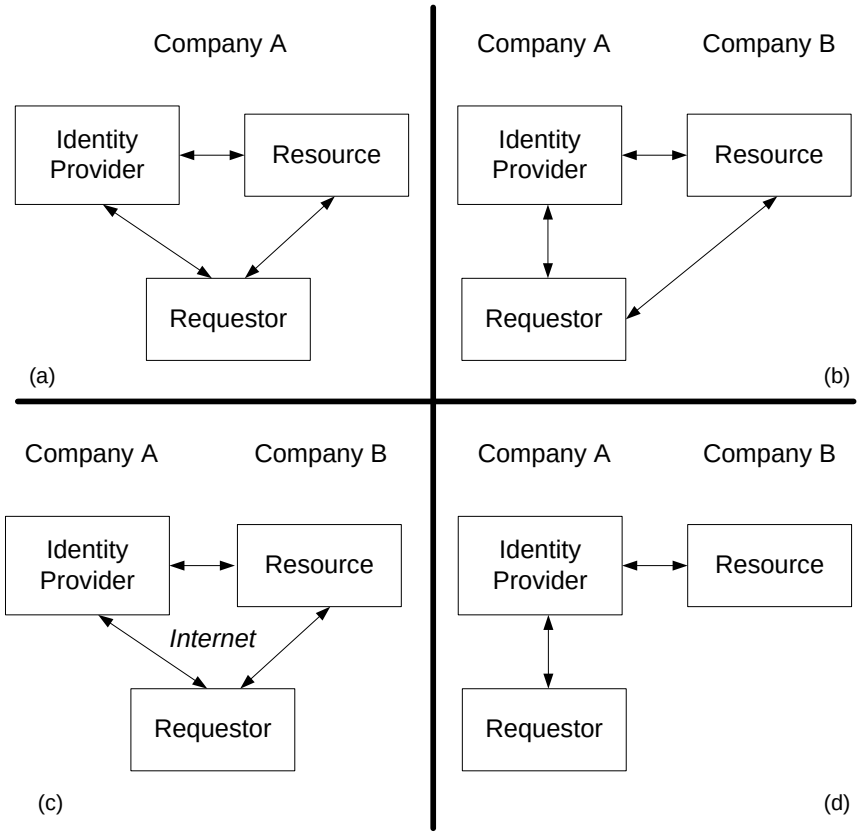
Federations must support a wide variety of configurations and environments. This framework leverages the WS-* specifications to create an evolutionary federation path allowing services to use only what they need and leverage existing infrastructures and investments.

Federations can exist within organizations and companies as well as across organizations and companies. They can also be ad-hoc collections of principals that choose to participate in a community. The figure below illustrates a few sample federations:



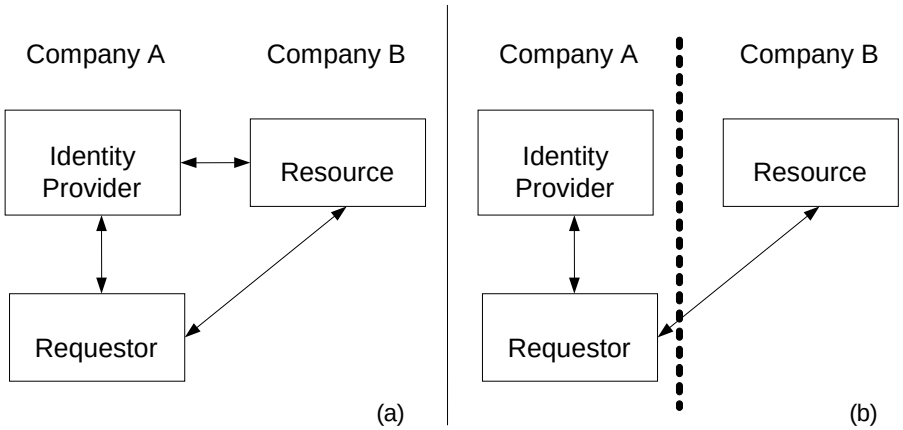
Figures 1a, 1b, 1c: Sample Federation Scenarios

As a consequence, federations MAY exist within one or multiple administrative domains, span multiple security domains, and MAY be explicit (requestor knows federation is occurring) or implicit (federation is hidden such as in a portal) as illustrated in the figure below:



Figures 2a, 2b, 2c, 2d: Sample Administrative Domains

Two points of differentiation for these models are the degree to which the Resource Provider and Identity Provider services can communicate and the levels of trust between the parties. For example, in cross-domain scenarios, the requestor's Identity Provider MAY be directly trusted and accessible or it MAY have a certificate from a trusted source and be hidden behind a firewall making it unreachable as illustrated in the Figure below:



Figures 3a, 3b: Accessibility of Identity Provider

In the federation process some level of information is shared. The amount of information shared is governed by policy and often dictated by contract. This is because the information shared is often of a personal or confidential nature. For example, this may indicate name, personal identification numbers, addresses, etc. In some cases the only information that is exchanged is an authentication statement (e.g. employee of company "A") allowing the actual requestor to be anonymous as in the example below:

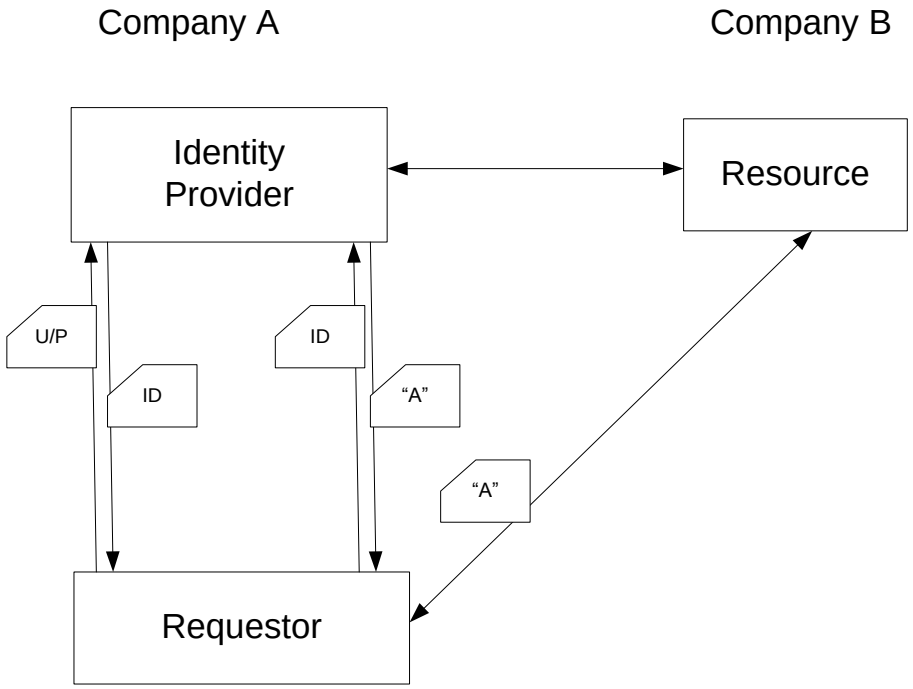


Figure 4: Sample Anonymous Access

To establish a federation context for a principal either the principal's identity is universally accepted (so that its association is "pre-established" across trust realms within a federation context), or it must be brokered into a trusted identity relevant to each trust realm within the federation context. The latter case requires the process of identity mapping – that is, the conversion of a digital identity from one realm to a digital identity valid in another realm by a party that trusts the starting realm and has the rights to speak for (make assertions

to) the ending realm, or make assertions that the ending realm trusts. Identity mapping (this brokering) is typically implemented by an IP/STS when initially obtaining tokens for a service or when exchanging tokens at a service's IP/STS.

A principal's digital identity can be represented in different forms requiring different types of mappings. For example, if a digital identity is fixed (immutable across realms within a federation), it may only need to be mapped if a local identity is needed. Fixed identities make service tracking (e.g. personalization) easy but this can also be a privacy concern (service collusion). This concern is lessened if the principal has multiple identities and chooses which to apply to which service, but collusion is still possible. Note that in some environments, collusion is desirable in that it can (for example) provide a principal with a better experience.

Another approach to identity mapping is pair-wise mapping where a unique digital identity is used for each principal at each target service. This simplifies service tracking (since the service is given a unique ID for each requestor) and prevents cross-service collusion by identity (if performed by a trusted service). While addressing collusion, this requires the principal's IP/STS to drive identity mapping.

A third approach is to require the service to be responsible for the identity mapping. That is, the service is given an opaque handle which it must then have mapped into an identity it understands – assuming it cannot directly process the opaque handle. More specifically, the requestor's IP/STS generates a digital identity that cannot be reliably used by the target service as a key for local identity mapping (e.g. the marker is known to be random or the marker's randomness is not known. The target service then uses the requestor's mapping service (called a pseudonym service) to map the given (potentially random) digital identity into a constant service-specific digital identity which it has registered with the requestor's mapping service. This also addresses the collusion issue but pushes the mapping burden onto the service (but keeps the privacy of all information in the requestor's control).

The following sections describe how the WS-* specifications are used and extended to create a federation framework to support these concepts.

2.2. Metadata Model

As discussed in the previous section, federations can be loosely coupled. As well, even within tightly coupled federations there is a need to discover the metadata and policies of the participants within the federation with whom a requestor is going to communicate.

This discovery process begins with the target service, that is, the service to which the requester wishes to ultimately communicate. Given the metadata endpoint reference (MEPR) for the target service allows the requestor to obtain all requirement metadata about the service (e.g. federation metadata, communication policies, WSDL, etc.).

This section describes the model where the MEPR points to an endpoint where the metadata can be obtain, which is, in turn, used to locate the actual service. An equally valid approach is to have a MEPR that points to the actual service and also contains all of the associated metadata (as described in [WS-MetadataExchange]) and thereby not requiring the extra discovery steps.

Federation metadata describes settings and information about how a service is used within a federation and how it participates in the federation. Federation metadata is only one component of the overall metadata for a service – there is also communication policy that describes the requirements for web service messages sent to the service and a WSDL description of the organization of the service, endpoints, and messages.

It should be noted that federation metadata, like communication policy, can be scoped to services, endpoints, or even to messages. As well, the kinds of information described are likely to vary depending on a services role within the federation (e.g. target service, security token service …).

Using the target service's metadata a requestor can discover the MEPRs of any related services that it needs to use if it is to fully engage with the target service. The discovery process is repeated for each of the related services to discover the full set of requirements to communicate with the target service. This is illustrated in the figure below:

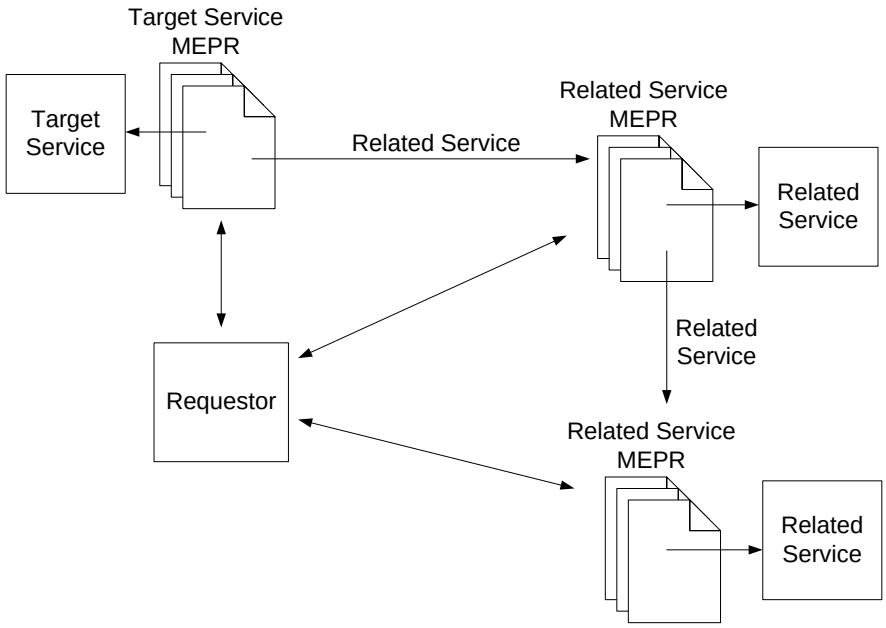


Figure 5a: Obtaining Federation Metadata (not embedded in EPR)

The discovery of metadata can be done statically or dynamically. Note that if it is obtained statically, there is a possibility of the data becoming stale resulting in communication failures.

As previously noted the MEPR MAY contain the metadata and refer to the actual service. That is, the EPR for the actual service MAY be within the metadata pointed to by the EPR (Figure 5a). As well, the EPR for the actual service MAY also contain (embed) the metadata (Figure 5b). An alternate view of Figure 5a in this style is presented in Figure 5b:

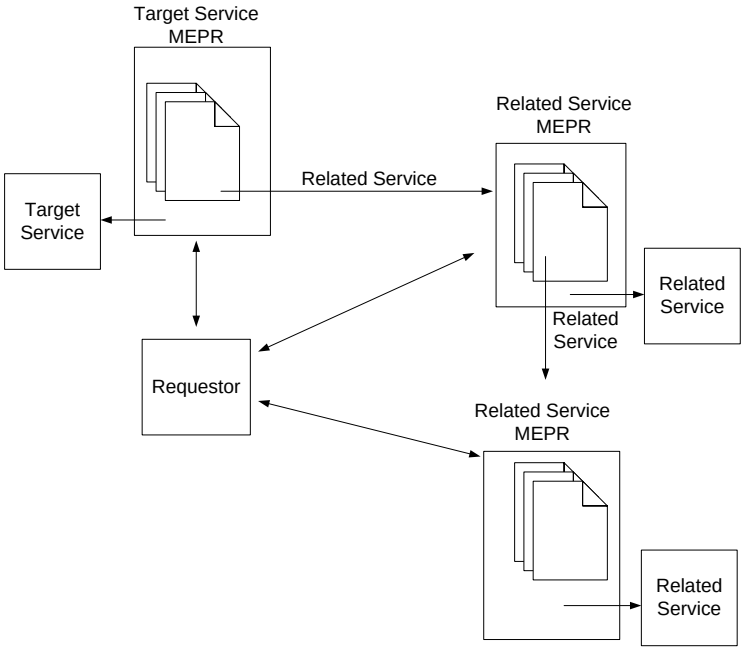


Figure 5b: Obtaining Federation Metadata (embedded)

Figures 5a and 5b illustrate homogenous use of MEPRs, but a mix is allowed. That is, some MEPRs might point at metadata endpoints where the metadata can be obtained (which contains the actual service endpoints) and some may contain actual service references with the service's metadata embedded within the EPR.

In some cases there is a need to refer to services by a name, thereby allowing a level of indirection to occur. This can be handled directly by the application if there are a set of well-known application-specific logical names or using some external mechanism or directory. In such cases the mapping of logical endpoints to physical endpoints is handled directly and such mappings are outside the scope of this specification. The following example illustrates the use of logical service names:

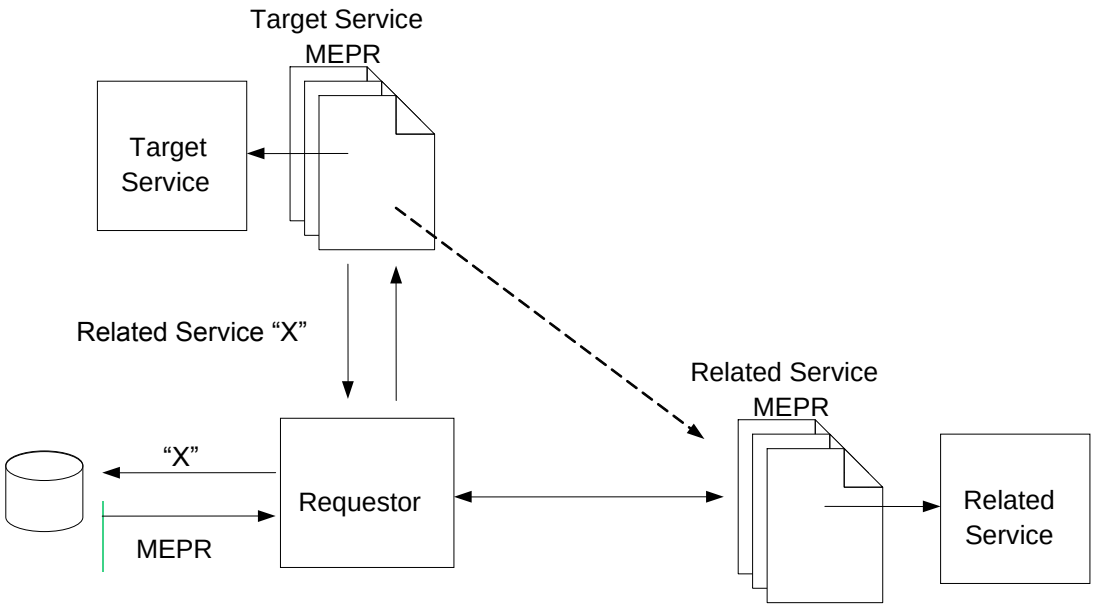


Figure 6: Example of Logical Service Names

To simplify metadata access, and to allow different kinds of metadata to be scoped to different levels of the services, both communication policies (defined in [WS-Policy]) and federation metadata (described in next chapter) can be embedded within WSDL using the mechanisms described in [WS-PolicyAttachment].

In some scenarios a service MAY be part of multiple federations. In such cases there is a need to make all federation metadata available, but there is often a desire to minimize what needs to be downloaded. For this reason federation metadata can reference metadata sections located elsewhere as well as having the metadata directly in the document. For example, this approach allows, a service to have a metadata document that has the metadata for the two most common federations in which the service participates and pointers (MEPR) to the metadata documents for the other federations. This is illustrated in the figure below:

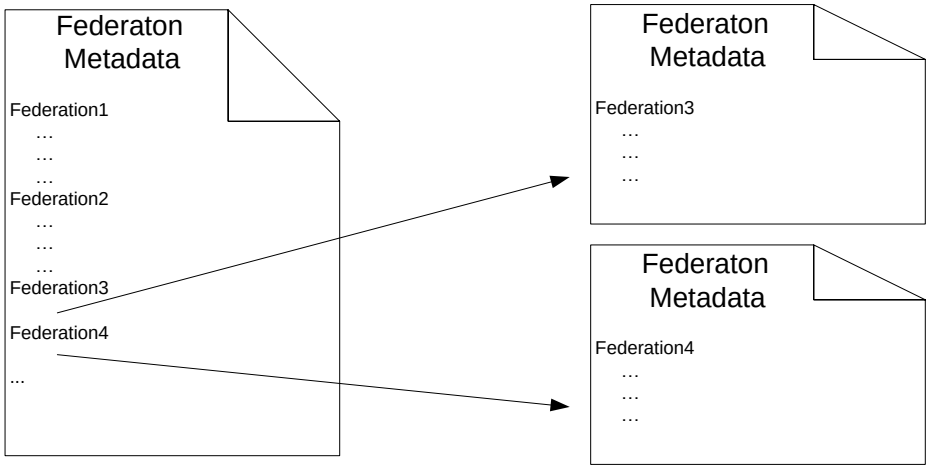


Figure 7: Federation Metadata Document

This section started by assuming knowledge of the MEPR for the target service. In some cases this is not known and a discovery process (described in section 3) is needed to obtain the federation metadata in order to bootstrap the process described in this section (e.g. using DNS or well-known addresses).

2.3. Security Model

As described in [WS-Trust], a web service MAY require a set of claims, codified in security tokens and related message elements, to process an incoming request. Upon evaluating the policy and metadata, if the requester does not have the necessary security token(s) to prove its right to assert the required claims, it MAY use the mechanisms described in [WS-Trust] (using security tokens or secrets it has already) to acquire additional security tokens.

This process of exchanging security tokens is typically bootstrapped by a requestor authenticating to an IP/STS to obtain initial security tokens using mechanisms defined in [WS-Trust]. Additional mechanisms defined in this specification along with [WS-MetadataExchange] can be used to enable the requestor to discover applicable policy, WSDL and schema about a service endpoint, which can in turn be used to determine the metadata, security tokens, claims, and communication requirements that are needed to obtain access to a resource (recall that federation metadata was discussed in the previous section).

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 exchange can be used to create a local access token or to map to a local identity.

This specification also describes an Attribute/Pseudonym service that can be used to provide mechanisms for restricted sharing of principal information and principal identity mapping (when different identities are used at different resources). The metadata mechanisms described in this document are used to enable a 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 or have an embedded IP/STS. Similarly, a requestor MAY also act as its own IP/STS or have an embedded IP/STS.

2.4. Trust Topologies and Security Token Issuance

The models defined in [WS-Security], [WS-Trust], and [WS-Policy] provides the basis for federated trust. This specification extends this foundation by describing how these models are combined to enable richer trust realm mechanisms across and within federations. This section describes different trust topologies and how token exchange (or mapping) can be used to broker the trust for each scenario. Many of the scenarios described in section 2.1 are illustrated here in terms of their trust topologies and illustrate possible token issuance patterns for those scenarios.

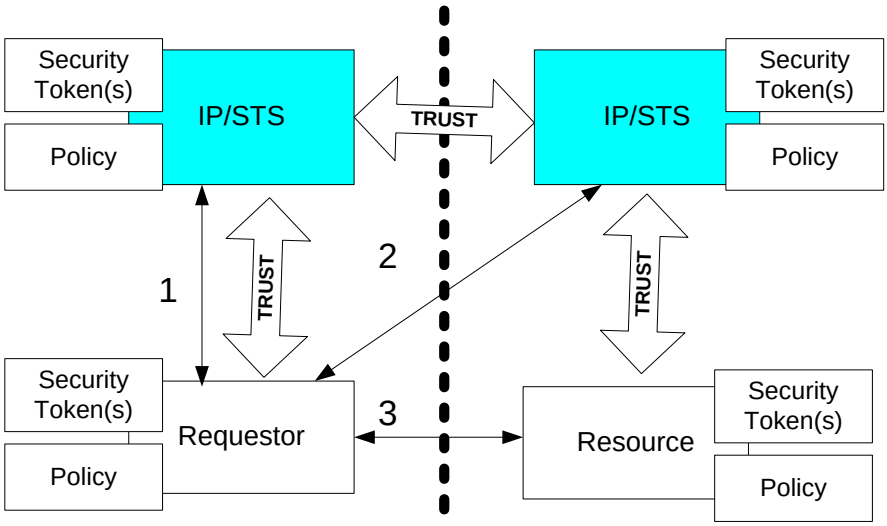


Figure 8: Federation and Trust Model

Figure 8 above illustrates one way the WS-Trust model may be applied to simple federation scenarios. Here security tokens (1) from the requestor's trust realm are used to acquire security tokens from the resource's trust realm (2) These tokens are then presented to the resource/service's realm (3) to access the resource/service . 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 (note that this process can be repeated allowing for trust chains of different lengths).

Note that in the figure above the trust of the requestor to its IP/STS and the resource to its IP/STS are illustrated. These are omitted from subsequent diagrams to make the diagrams for legible.

Figure 9 below illustrates another approach where the resource/service acts as a validation service. In this scenario, the requestor presents the token provided by the requestor's STS (1, 2) to the resource provider, where the resource provider uses its security token service to understand and validate this security token(s) (3). In this case information on the validity of the presented token should be returned by the resource provider's token service.

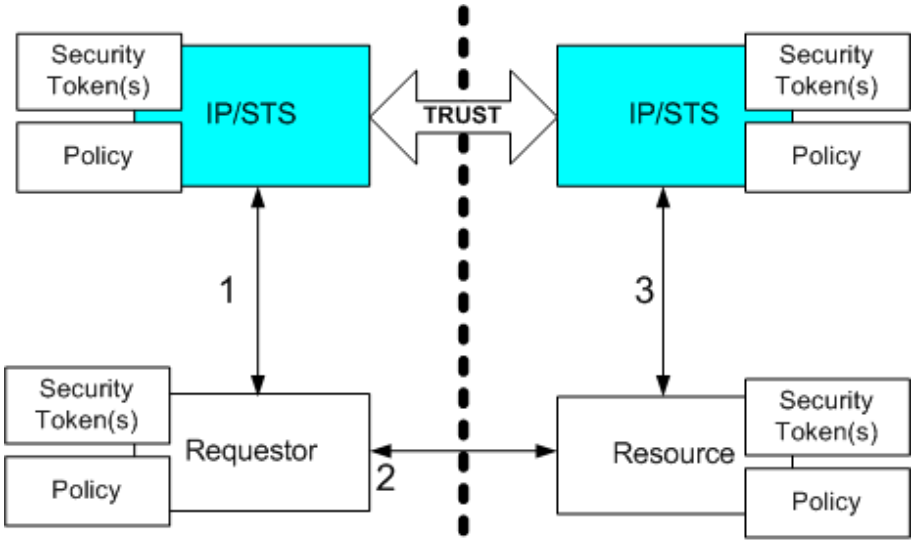


Figure 9: Alternate Federation and Trust Model

Note that the model above also allows for different IP/STS services within the same trust realm (e.g. authentication and authorization services).

In both of the above examples, a trust relationship has been established between the security token services. Alternatively, as illustrated in Figure 10, there may not be a direct trust relationship, but an indirect trust relationship that relies on a third-party to establish and confirm separate direct trust relationships.

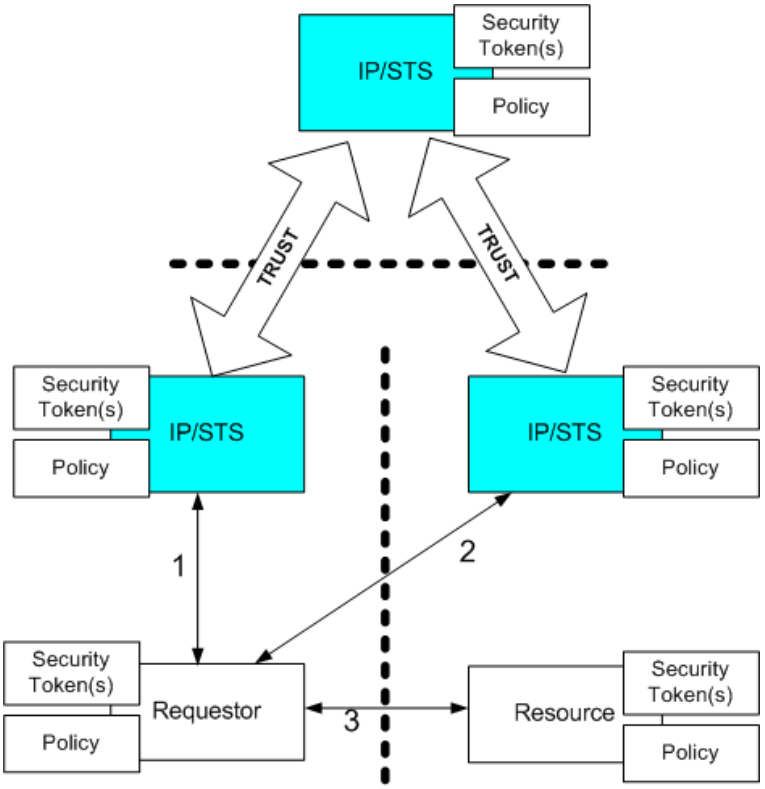


Figure 10: Indirect Trust

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:

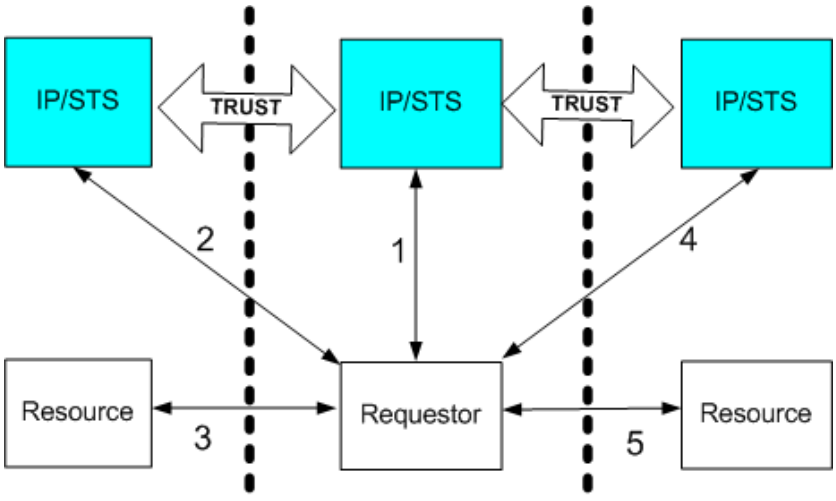


Figure 11: Multiple Trust Domains

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 12:

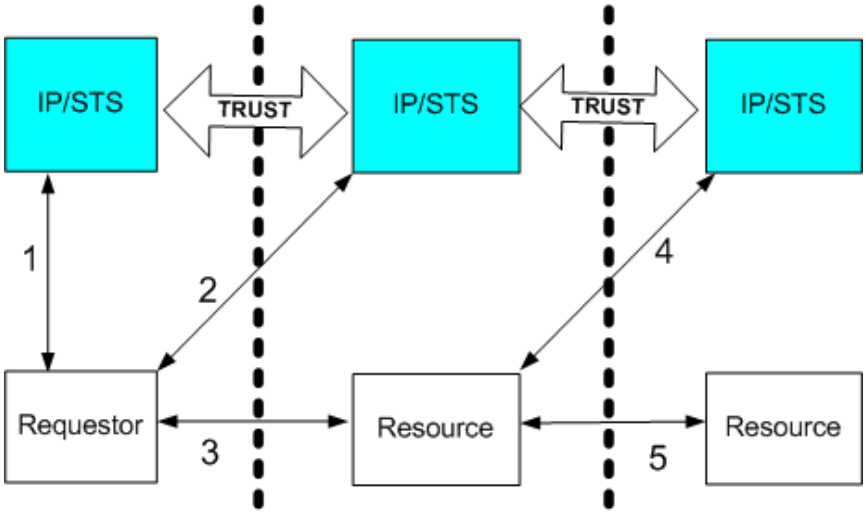


Figure 12: Trust between Requestor-Resource and Resource-Delegate Resource

In such cases (as illustrated in Figure 12) the first resource, in its capacity as a second requestor on behalf of the original requestor, provides security tokens to allow/indicate proof of (ability for) 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.

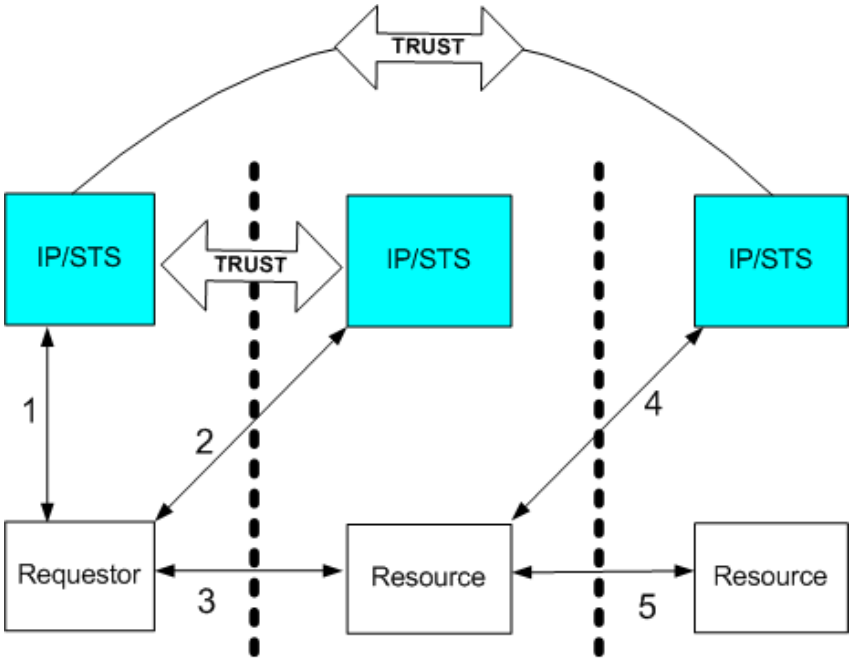


Figure 13: No Trust Relationship between Resource Providers

Specifically, in Figure 13 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 `<wst:RequestSecurityToken>` request.

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

In all cases, the original requestor 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, or by the service's policy.

Another form of federation involves *ad hoc* networks of *peer trust*. That is, there may be direct trust relationships that are not based on certificate chains. In such cases an identity's chain is irrelevant or may even be self-signed. Such trusts may be enforced at an IP/STS or at a Relying Party directly.

2.5. 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.

One possible function of a security token service is to provide digital identities – an *Identity Provider (IP)*. This is a special type of security token service that, at a minimum, performs authentication and can make identity (or origin) 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 Identity Provider (IP) and STS. In Figure 14, a requestor obtains an identity security token from its Identity Provider (1) 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 then uses the access token on requests to the resource or Web service (3). Note that it is assumed that there is a trust relationship between the STS and the identity provider.

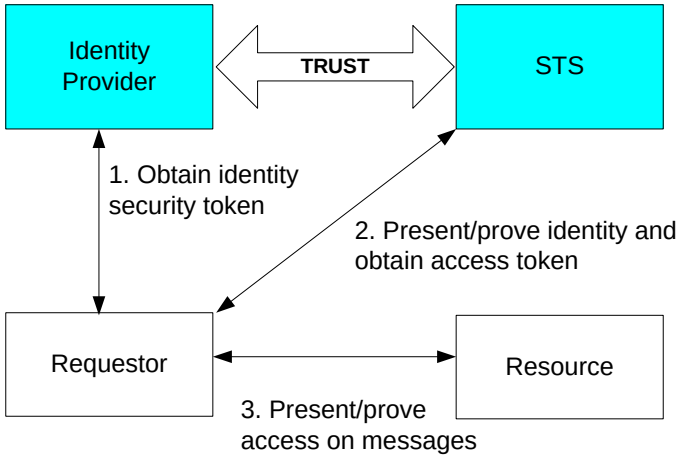


Figure 14: Role of IP/STS in Basic Federation Model

2.6. Attributes and Pseudonyms

Attributes are typically used when applications need additional information about the requestor that has not already been provided or cached, or is not appropriate to be sent in

every request or saved in security tokens. Attributes are also used when ad hoc information is needed that cannot be known at the time the requests or token issuance.

Protecting privacy in a federated environment often requires additional controls and mechanisms. One such example is detailed access control for 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 or mapping of separately managed identities.

When requestors interact with resources in different trust realms (or different parts of a federation), there is often a need to *know* additional information about the requestor in order to authorize, process, or personalize the experience. A service, known as an *Attribute Service* may be available within a realm or federation. As such, an attribute service is used to provide the attributes about a requestor that are relevant to the completion of a request, given that the service is authorized to obtain this information. This approach 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 principal needs to be 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. While some scenarios support identities that are trusted as presented, pseudonyms services allow those cases where identity mapping needs to occur between an identity and a pseudonym on behalf of the principal.

There are different approaches to identity mapping. For example, the mapping can be performed by the IP/STS when requesting a token for the target service. Alternatively, target services can register their own mappings. This latter approach is needed when the digital identity cannot be reliability used as a key for local identity mapping (e.g. when a random digital identity is used not a constant or pair-wise digital identity).

Figure 15 illustrates the general model for Attribute & Pseudonym Services (note that there are different variations which are discussed later in this specification). This figure illustrates two realms with associated attribute/pseudonym services and some of the possible interactions. Note that it is assumed that there is a trust relationship between the realms.

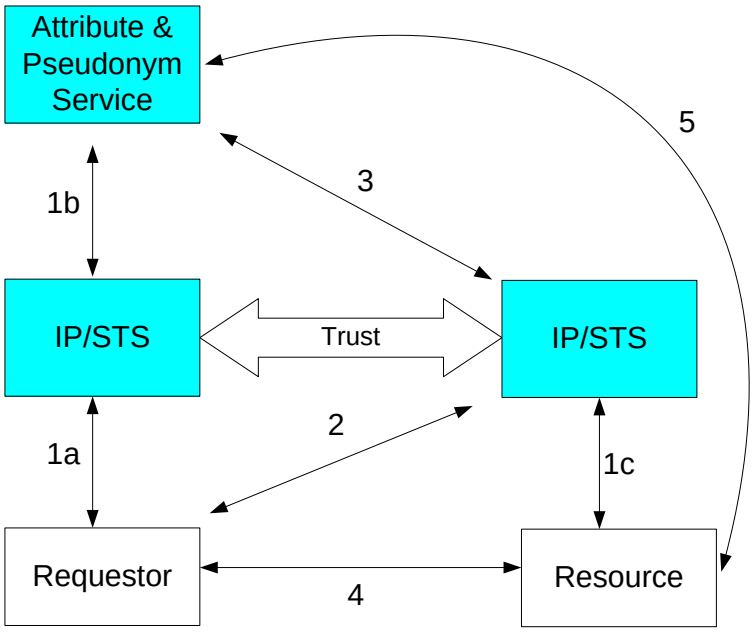


Figure 15: Attributes & Pseudonyms

With respect to Figure 15, 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) to obtain an access token for the resource. In this example the resource IP/STS has registered a pseudonym with the requestor's pseudonym service (3) possibly for sign-out notification or for service-driven mappings. The requestor accesses the resource using the pseudonym token (4). The resource can obtain additional 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). Note that if the mapping occurs at the IP/STS then a service-consumable identity is returned in step 1a.

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, while protecting the principal’s privacy. 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. The need for service-driven mapping is typically known up-front or indicated in metadata.

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 providers 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 providers issue a constant digital identity such as a name or ID at a particular realm. However, there is often a desire to prevent identity collusion between service providers. This specification provides two possible countermeasures. The first approach is to have identity providers issue random (or pseudo-random, pair wise, etc.) IDs each time a requestor signs in. This means that the resulting identity token contains a unique (or relatively unique) identifier, typically random, that hides their identity. As such, it cannot be used (by itself) as a digital identity (e.g. for personalization). The identity needs to be mapped into a service-specific digital identity. This can be done by the requestor ahead of time when requesting a service-specific token or by the service when processing the request. The following example illustrate mapping by the service.

In this example the unique identity returned 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.

A variation on this first approach is the use of randomly generated pseudonyms; 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:

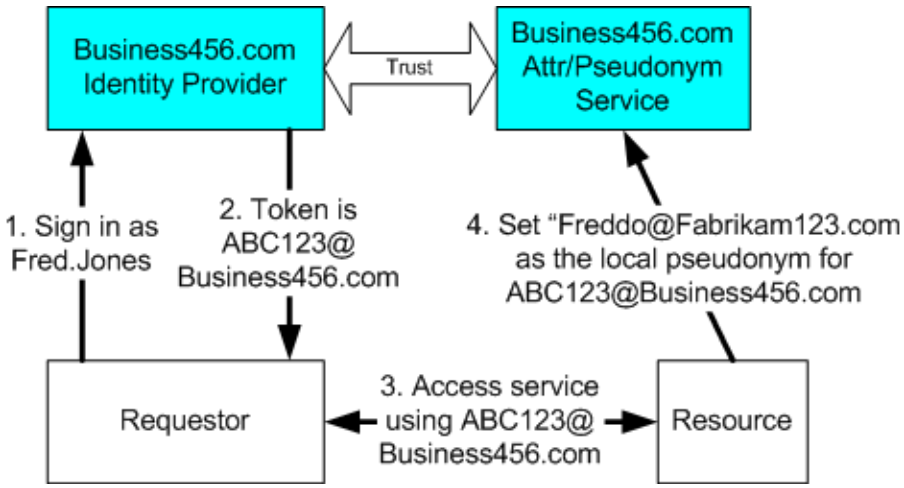


Figure 16: Pseudonym

Note that the attribute, pseudonym, and Identity Provider services could be combined or separated in many different configurations. Figure 16 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, it might return a new identifier, like XYZ321@Business456.com, in the token to be presented to Fabrikam in step 3. The Web service at Fabrikam123.com can now request a local pseudonym for XYZ321@Business456.com and be told "Freddo@Fabrikam123.com" This is possible because the Business456 pseudonym service interacts with the Business456 IP and is authorized and allowed under the principal's privacy policy to reverse map "XYZ321@Business456.com" into a known identity at Business456.com which has associated with it pseudonyms for different realms. (Note that later in this section a mechanism for directly returning the pseudonym by the IP is discussed). Figure 17 below illustrates this scenario:

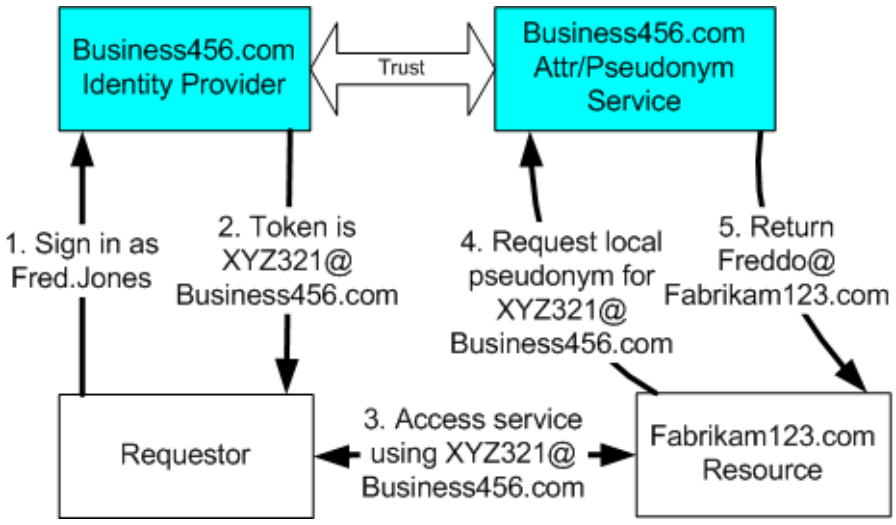


Figure 17: Pseudonym - local id

Now the Fabrikam web service can complete the request using the local name to obtain data stored within the local realm on behalf of the requestor as illustrated below:

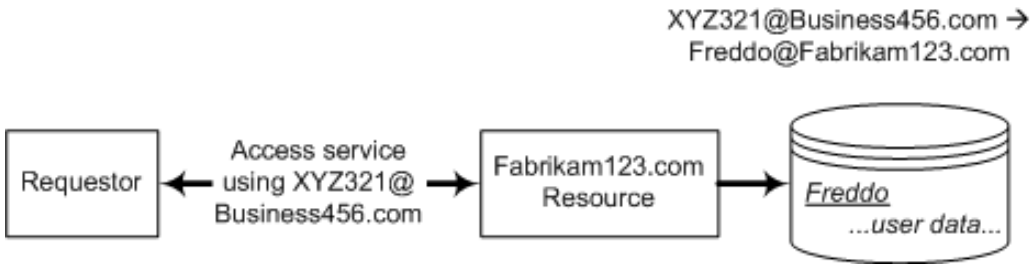


Figure 18: Pseudonym - local realm

Another variation of the first approach is to have the requestor map the identity, by creating pseudonyms for specific services. In this case 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 19 below:

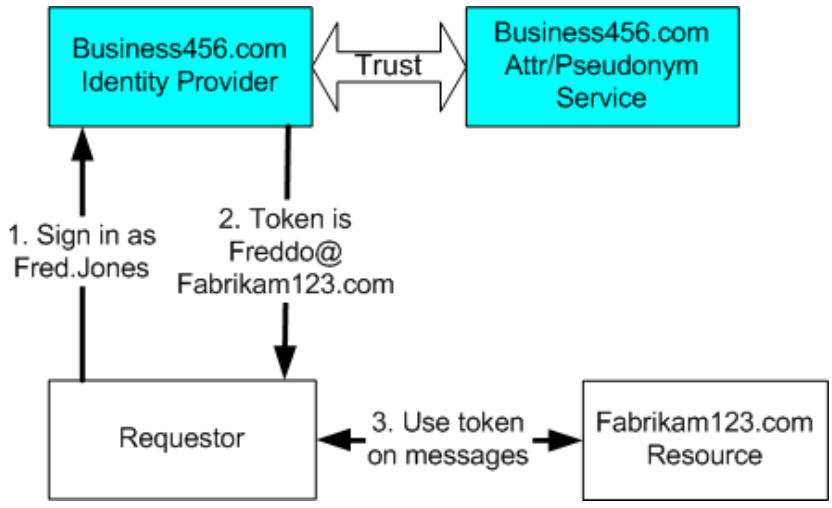


Figure 19: Pseudonym – token acceptance

The second approach is to create static identities for each service (or a group of services). That is, principle A at service X is given the digital identity 12, principle A at service Y is given the digital identity 75, principle B at service X is given the digital identity 46, and so on. Operationally this approach is much like the last variation from the first approach. That is, the requestor must map its identity to an identity for the service (or service group) via a token request from its IP/STS (or using the pseudonym service directly). Consequently requestor mapping from random identities and pair-wise mapping are functionally equivalent.

2.7. Attributes, Pseudonyms, and IP/STS Services

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 and attribute retrieval mechanisms, while protecting a principals' privacy. Figure 20 below illustrates the key aspects of this extended model:

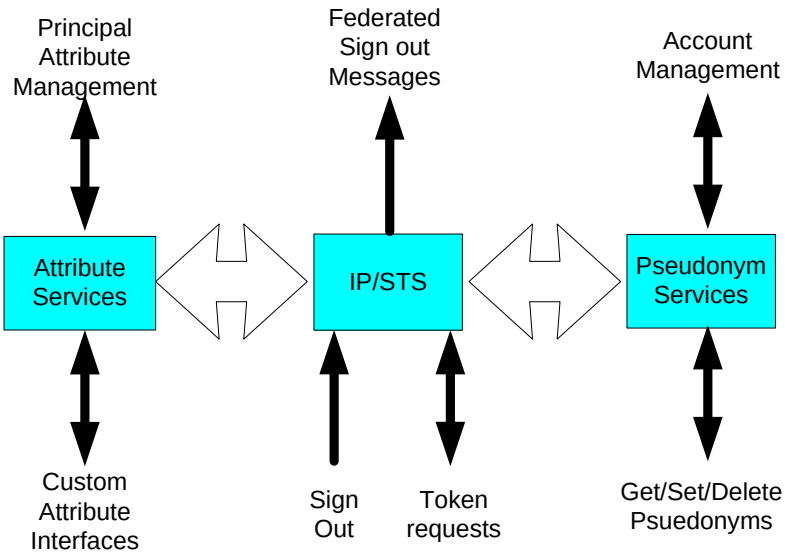


Figure 20: Pseudonyms, Attributes and Token Issuance

As shown above, Principals request security tokens from Identity Providers and security token services. As well, Principals MAY send sign-out requests (either explicitly as described later or implicitly by cancelling tokens) 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) optionally sends sign-out notifications to subscribers (as described later). Principals are associated with the attribute/pseudonym services and attributes and pseudonyms are added and used.

3. Federation Metadata

Once two parties have made the decision to federate their computing systems, it is usually necessary to configure their respective systems to enable federated operation. For example, the officers of a company such as contoso.com might reach a business arrangement where they choose to provide a set of services to someone who can present

identity credentials (in the form of security tokens) issued by fabrikam.com. In this example, it may be necessary for contoso.com administrator to update a local database with the public key that fabrikam.com uses to sign its security tokens. In addition to the signing key, it may be necessary for an organization to make available other types of information pertinent to a federated relationship. Depending on the arrangement between the organizations, in some cases it is desirable to help automate this configuration process.

This section defines a XML document format for *federation metadata* that can be made available by an organization to make it easier for partners to federate with that organization. Furthermore, this section defines a process by which this document can be obtained securely.

It should be noted that a service may be part of multiple federations and be capable of receiving messages at the same endpoint in the context of all, or some subset of these federations. Consequently the federation metadata document allows for statements to be made about each federation.

The metadata document can take different forms. The following list identifies a few common forms:

- A document describing the metadata for a single federation
- A document with separate sections for each federation, when a service is part of multiple federations
- A document with references to metadata documents
- A document for a single service identifying multiple issuance MEPRs that are offered by the service (the MEPRs can be used to obtain issuer-specific metadata)
- A document embedded inside of a WSDL description (described below)

Federation metadata documents may be obtained in a variety of ways as described in section 3.2. It should be noted that services MAY return different federation metadata documents based on the identity and claims presented by a requestor.

3.1 Federation Metadata Document

The federation metadata document is a container that organizations can fill to proffer information that may be useful to partners for establishing a federation. This section defines the overall document format and several optional elements that MAY be included in the federation metadata document.

The federation metadata document MUST be of the following form:

```
<?xml version="1.0" encoding="..." ?>
<fed:FederationMetadata xmlns:fed="..." ...>
  <fed:Federation [FederationID="..."] ...> +
    [Federation Metadata]
  </fed:Federation>
 [Signature]
</fed:FederationMetadata>
```

The document consists of one or more *federation* sections which describe the metadata for the endpoint within a federation. The federation section MAY specify an URI indicating an identifier for the federation using the `FederationID` attribute, or it MAY omit this identifier indicating the "default federation". However, only one section may omit the FederationID attribute if multiple sections are provided.

The **[Federation Metadata]** property of the metadata document represents a set of one or more optional XML elements within a federation scope that the federation metadata provider wants to supply to its partners. The **[Signature]** property provides a digital signature (typically using XML Digital Signature [XML-Signature]) over the federation metadata document to ensure data integrity and provide data origin authentication. The recipient of a federation metadata document SHOULD ignore any metadata elements that it does not understand or know how to process.

Participants in a federation have different roles. Consequently not all metadata statements apply to all roles. There are three general roles: requestors who make web service requests, security token services who issues federated tokens, and service provides who rely on tokens from token providers.

The following table outlines the common roles and associated metadata statements:

Role	***Applicable Metadata Statements***
Any participant	mex:MetadataReference, fed:AttributeServiceEndpoint
Security Token Service	fed:TokenSigningKeyInfo, fed:PseudonymServiceEndpoint, fed:SingleSignOutSubscriptionEndpoint, fed:TokenTypesOffered, fed:UriNamedClaimTypesOffered, fed:AutomaticPseudonyms fed:IssuerNamesOffered
Service provider / Relying Party (includes Security Token Service)	fed:TokenIssuerName, fed:TokenIssuerEndpoint fed:TokenKeyTransferKeyInfo, fed:SingleSignoutNotificationEndpoint

The contents of the federated metadata are extensible so services can add new elements. Each federated metadata statement MUST define if it is optional or required for specific roles. When processing a federated metadata document, unknown elements SHOULD be ignored.

The following sections detail referencing federation metadata documents, the predefined elements, signing metadata documents, and provide a sample federation metadata document.

3.1.1. Referencing Other Metadata Documents

An endpoint MAY choose not to provide the statements about each federation to which it belongs. Instead it MAY provide an endpoint reference to which a request for federation metadata can be sent to retrieve the metadata for that specific federation. This is indicated by placing a `<mex:MetadataReference>` element inside the `<fed:Federation>` for the federation. In such cases the reference MUST identify a document containing only federation metadata sections. Retrieval of the referenced federation metadata documents is done using the mechanisms defined in [WS-MetadataExchange]. The content MUST match the reference context. That is, if the reference is from the default `<fed:Federation>` then the target MUST contain a `<fed:FederationMetadata>` document with a default `<fed:Federation>`. If the reference is from a `<fed:Federation>` element with a FederationID then the target MUST contain a `<fed:FederationMetadata>` document with a

<fed:Federation> element that has the same FederationID as the source <fed:Federation> element.

It should be noted that an endpoint MAY choose to only report a subset of federations to which it belongs to requestors.

The following pseudo-example illustrates a federation metadata document that identifies participation in three federations. The metadata for the default federation is specified in-line within the document itself, whereas metadata references are specified for details on the other two federations.

```
<?xml version="1.0" encoding="utf-8" ?>
<fed:FederationMetadata xmlns:fed="..."
                        xmlns:mex="..."
                        xmlns:wsa="..."
                        xmlns:wsse="..."
                        xmlns:ds="...">
  <fed:Federation>
    <fed:TokenSigningKeyInfo>
      <wsse:SecurityTokenReference>
        <ds:X509Data>
         <ds:X509Certificate>
            ...
         </ds:X509Certificate>
       </ds:X509Data>
      </wsse:SecurityTokenReference>
    </fed:TokenSigningKeyInfo>
    ...
  </fed:Federation>
  <fed:Federation FederationID="http://example.com/federation35532">
    <mex:MetadataReference>
      <wsa:Address>http://example.com/federation35332/FedMD
      </wsa:Address>
    </mex:MetadataReference>
  </fed:Federation>
  <fed:Federation FederationID="http://example.com/federation54478">
    <mex:MetadataReference>
      <wsa:Address>http://example.com/federation54478/FedMD
      </wsa:Address>
    </mex:MetadataReference>
  </fed:Federation>
</fed:FederationMetadata>
```

Federation metadata documents can also be named with a URI and referenced to allow sharing of content (e.g. at different endpoints in a WSDL file). To share content between two <fed:Federation> elements the <fed:FederationInclude> element is used. When placed inside a <fed:Federation> element the <fed:FederationInclude> element indicates that the identified federation's metadata statements are effectively copied into the containing <fed:Federation> element.

For example, the following examples are functionally equivalent:

```
<?xml version="1.0" encoding="utf-8" ?>
<fed:FederationMetadata  xmlns:fed="..." xmlns:wsse="..." xmlns:ds="...">
  <fed:Federation FederationID="http://example.com/f1">
    <fed:TokenSigningKeyInfo>
      <wsse:SecurityTokenReference>
        <ds:X509Data>
         <ds:X509Certificate>
            ...
          </ds:X509Certificate>
        </ds:X509Data>
      </wsse:SecurityTokenReference>
    </fed:TokenSigningKeyInfo>
  </fed:Federation>
  <fed:Federation FederationID="http://example.com/federation35532">
    <fed:TokenSigningKeyInfo>
      <wsse:SecurityTokenReference>
        <ds:X509Data>
         <ds:X509Certificate>
            ...
          </ds:X509Certificate>
        </ds:X509Data>
      </wsse:SecurityTokenReference>
    </fed:TokenSigningKeyInfo>
  </fed:Federation>
</fed:FederationMetadata>
```

and

```
<?xml version="1.0" encoding="utf-8" ?>
<fed:FederationMetadata  xmlns:fed="..." xmlns:wsse="..." xmlns:ds="...">
  <fed:Federation FederationID="http://example.com/f1">
    <fed:TokenSigningKeyInfo>
      <wsse:SecurityTokenReference>
        <ds:X509Data>
         <ds:X509Certificate>
            ...
          </ds:X509Certificate>
        </ds:X509Data>
      </wsse:SecurityTokenReference>
    </fed:TokenSigningKeyInfo>
  </fed:Federation>
  <fed:Federation FederationID="http://example.com/federation35532">
    <fed:FederationInclude>http://example.com/f1</fed:FederationInclude>
  </fed:Federation>
</fed:FederationMetadata>
```

Typically a <fed:FederationInclude> reference identifies a <fed:Federation> element elsewhere in the document. However, the URI MAY represent a "well-known" metadata document that is known to the processor. The mechanism by which a processor "knows" such URIs is undefined and outside the scope of this specification.

When referencing or including other metadata documents the contents are logically combined. As such it is possible for some elements to be repeated. While the semantics of this is defined by each element, typically it indicates a union of the information. That is, both elements apply.

The mechanisms defined in this section allow creation of composite federation metadata documents. For example, if there is metadata common to multiple federations it can be described separately and then referenced from the definitions of each federation which can then include additional (non-conflicting) metadata specific to the federation.

3.1.2 TokenSigningKeyInfo Element

The optional <fed:TokenSigningKeyInfo> element allows a federation metadata provider to specify what key will be used by it to sign security tokens issued by it. This is only specified by token issuers and security token services. This is typically a service-level statement but can be an endpoint-level statement. This element populates the [Federation Metadata] property. The signing key can be specified using any of the mechanisms supported by the <wsse:SecurityTokenReference> element defined in [WS-Security] as shown below.

```
<fed:TokenSigningKeyInfo ...>
  <wsse:SecurityTokenReference>
    ...
  </wsse:SecurityTokenReference>
</fed:TokenSigningKeyInfo>
```

Any top-level element legally allowed as a child of the ds:KeyInfo element (as per [XML-Signature]) can appear as a child of the <wsse:SecurityTokenReference> element.

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

For example, the token signing key can be carried inside an X.509 certificate and specified as follows.

```
<fed:TokenSigningKeyInfo>
  <wsse:SecurityTokenReference>
    <ds:X509Data>
      <ds:X509Certificate>
MIIBsTCCAV+gAwIBAgIQz9jmro9+5ahJyMQzgtSAvzAJBgUrDgMCHQUAMBYxFDASBgNVBAMTC1Jvb
3QgQWdlbmN5MB4XDTA1MDkwMTExNTUzNFoXDTM5MTIzMTIzNTk1OVowFDESMBAGA1UEAxMJbG9jYW
xob3N0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCnK1hCowhf6K3YrKoKuB87j6rdCrSHrne
xzkPeg1YDwp6GquI3DVaD+VNRySREnI1yrqjDWyprp4FiJesPgs94PJRE6wz6Y5Z1CfhMUslh2t+X
hBtJycvmLEZX+3Lt2y6PCf49qlwfX/TqReCiMKYM9h+OVN32sFPQnz6dMUfH4QIDAQABo0swSTBHB
gNVHQEEQDA+gBAS5AktBh0dTwCNYSHcFmRjoRgwFjEUMBIGA1UEAxMLUm9vdCBBZ2VuY3mCEAY3bA
CqAGSKEc+41KpcNfQwCQYFKw4DAh0FAANBAFLLkisG9ojZ2QtIfwjVJUdrsNzBO8JZOrLl8lZd9I/
/hZ6643L4sblBFB8ttbJjT4rdt5sKjpezRn3ZVIcvbQE=
      </ds:X509Certificate>
    </ds:X509Data>
  </wsse:SecurityTokenReference>
</fed:TokenSigningKeyInfo>
```

Note that an X.509 certificate chain can also be specified using this mechanism since the `ds:X509Data` element supports specifying a chain. There are no requirements that the signing key be a leaf certificate – it can be anywhere in a certificate chain.

As another example, the token signing key can be specified as a raw RSA key as follows.

```
<fed:TokenSigningKeyInfo>
  <wsse:SecurityTokenReference>
    <ds:RSAKeyValue>
      <ds:Modulus>
A7SEU+e0yQH5rm9kbCDN9o3aPIo7HbP7tX6WOocLZAtNfyxSZDU16ksL6WjubafOqNEpcwR3RdFsT
7bCqnXPBe5ELh5u4VEy19MzxkXRgrMvavzyBpVRgBUwUlV5foK5hhmbktQhyNdy/6LpQRhDUDsTvK
+g9Ucj47es9AQJ3U=
      </ds:Modulus>
      <ds:Exponent>AQAB</ds:Exponent>
    </ds:RSAKeyValue>
  </wsse:SecurityTokenReference>
</fed:TokenSigningKeyInfo>
```

3.1.3 TokenKeyTransferKeyInfo Element

The optional `<fed:TokenKeyTransferKeyInfo>` element allows a federation metadata provider, a security token service or Relying Party in this case, to specify what key should be used to encrypt keys and key material targeted for the service. This is typically a service-level statement but can be an endpoint-level statement. This element populates the [Federation Metadata] property. The key transfer key can be specified using any of the mechanisms supported by the `<wsse:SecurityTokenReference>` element defined in [WS-Security] as shown below.

```
<fed:TokenKeyTransferKeyInfo ...>
  <wsse:SecurityTokenReference>
    ...
  </wsse:SecurityTokenReference>
</fed:TokenKeyTransferKeyInfo>
```

Any top-level element legally allowed as a child of the `ds:KeyInfo` element (as per [XML-Signature]) can appear as a child of the `<wsse:SecurityTokenReference>` element.

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

For example, the token signing key can be carried inside an X.509 certificate and specified as follows.

```
<fed:TokenKeyTransferKeyInfo>
  <wsse:SecurityTokenReference>
    <ds:X509Data>
      <ds:X509Certificate>
MIIBsTCCAV+gAwIBAgIQz9jmro9+5ahJyMQzgtSAvzAJBgUrDgMCHQUAMBYxFDASBgNVBAMTC1Jvb
3QgQWdlbmN5MB4XDTA1MDkwMTExNTUzNFoXDTM5MTIzMTIzNTk1OVowFDESMBAGA1UEAxMJbG9jYW
xob3N0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCnK1hCowhf6K3YrKoKuB87j6rdCrSHrne
xzkPeg1YDwp6GquI3DVaD+VNRySREnI1yrqjDWyprp4FiJesPgs94PJRE6wz6Y5Z1CfhMUslh2t+X
hBtJycvmLEZX+3Lt2y6PCf49qlwfX/TqReCiMKYM9h+OVN32sFPQnz6dMUfH4QIDAQABo0swSTBHB
gNVHQEEQDA+gBAS5AktBh0dTwCNYSHcFmRjoRgwFjEUMBIGA1UEAxMLUm9vdCBBZ2VuY3mCEAY3bA
CqAGSKEc+41KpcNfQwCQYFKw4DAh0FAANBAFLLkisG9ojZ2QtIfwjVJUdrsNzBO8JZOrLl8lZd9I/
/hZ6643L4sblBFB8ttbJjT4rdt5sKjpezRn3ZVIcvbQE=
```

```
      </ds:X509Certificate>
    </ds:X509Data>
  </wsse:SecurityTokenReference>
</fed:TokenSigningKeyInfo>
```

Note that if this element isn't specified, and the signing key doesn't prohibit key transfer, it MAY be used as the key transfer key.

3.1.4 IssuerNamesOffered Element

In some scenarios token issuers are referred to be a logical name representing an equivalence class of issuers. For example, a Relying Party may not care what specific bank issues a token so long as the issuance is associated with a specific credit card program. To facilitate this, federated metadata provides the `<fed:IssuerNamesOffered>` element to indicate the logical names associated with an issuer and the `<fed:TokenIssuerName>` element (described below) to indicate that a Relying Party needs a token from a specific class of issuer.

As stated, the optional `<fed:IssuerNamesOffered>` element allows a federation metadata provider, specifically a token service in this case, to specify a set of "logical names" that are associated with the provider. That is, when a Relying Party indicates a logical name for a token issuer using the `<fed:TokenIssuerName>` element this element can be used as a correlation mechanism by clients. This element populates the [Federation Metadata] property. This is typically a service-level statement but can be an endpoint-level statement.

The schema for this optional element is shown below.

```
<fed:IssuerNamesOffered ...>
  <fed:IssuerName Uri="xs:anyURI" .../> +
</fed:IssuerNamesOffered>
```

The following example illustrates using this optional element to specify a logical name of the federating organization as a token issuer.

```
<fed:IssuerNamesOffered>
  <fed:IssuerName Uri="http://fabrikam.com/federation/corporate" />
</fed:IssuerNamesOffered>
```

3.1.5 TokenIssuerName Element

The optional `<fed:TokenIssuerName>` element allows a federation metadata provider to specify a "logical name" for a trusted token issuer or class of issuer which can be used by a client when requesting tokens to be consumed by the metadata provider (see section 3.1.4 for an overview of the mechanism). More specifically, the logical name references the metadata provider organization as an issuer entity. This element populates the [Federation Metadata] property. This is specified by any Relying Party (e.g. token issuers, security token services, and service providers). This is typically a service-level statement but can be an endpoint-level statement. This element MAY be specified even if the `<fed:TokenIssuerEndpoint>` element is specified.

The schema for this optional element is shown below.

```
<fed:TokenIssuerName ...> xs:anyURI </fed:TokenIssuerName>
```

The content of this element is an URI which serves as a logical name of the issuer entity. This logical name does not say or imply anything about the specific endpoint(s) of an STS

that can issue corresponding federated tokens. The issuing STS endpoints must be either explicitly specified (see Section 3.1.6) in the metadata or resolved by some other means.

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

The following example illustrates using this optional element to specify a logical name of the federating organization as a token issuer.

```
<fed:TokenIssuerName>
  http://fabrikam.com/federation/corporate
</fed:TokenIssuerName>
```

3.1.6 TokenIssuerEndpoint Element

The optional `<fed:TokenIssuerEndpoint>` element allows a federation metadata provider to specify the endpoint address a trusted STS which can be used by federated partners when requesting tokens to be consumed by the metadata provider. This element populates the [Federation Metadata] property. This is specified by any Relying Party (e.g. token issuers, security token services, and service providers). This is typically a service-level statement but can be an endpoint-level statement. This element MAY be specified even if the `<fed:TokenIssuerName>` element is specified.

The schema for this optional element is shown below.

```
<fed:TokenIssuerEndpoint>
  wsa:EndpointReferenceType
</fed:TokenIssuerEndpoint>
```

The content of this element is an endpoint reference as defined by [WS-Addressing] that identifies an endpoint address for the issuer STS. The endpoint reference MAY (and SHOULD if there is no expectation that the policy is known *a priori*) include metadata for the STS endpoint or a reference to an endpoint from where such metadata can be retrieved by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for additional details).

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

It should be noted that this element MAY occur multiple times indicating different services or different endpoints at a common service.

The following example illustrates using this optional element to specify an endpoint address for the token issuing STS of the federating organization.

```
<fed:TokenIssuerEndpoint>
  <wsa:Address> http://fabrkam.com/federation/STS </wsa:Address>
</fed:TokenIssuerEndpoint>
```

3.1.7 PseudonymServiceEndpoint Element

The optional `<fed:PseudonymServiceEndpoint>` element allows a federation metadata provider to specify the endpoint address of its pseudonym service which can be referenced by federated partners when requesting tokens from it. When present, this indicates that services SHOULD use the pseudonym service to map identities to local names as the identities MAY vary across invocations. This element populates the [Federation Metadata] property. This is typically specified by token issuers and security token services. This is typically a service-level statement but can be an endpoint-level statement.

The schema for this optional element is shown below.

```
<fed:PseudonymServiceEndpoint>
```

```
  wsa:EndpointReferenceType
</fed:PseudonymServiceEndpoint>
```

The content of this element is an endpoint reference as defined by [WS-Addressing] providing a transport address for the pseudonym service. The endpoint reference MAY (and SHOULD if there is no expectation that the policy is known *a priori*) include metadata for the STS endpoint or a reference to an endpoint from where such metadata can be retrieved by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for additional details).

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

It should be noted that this element MAY occur multiple times indicating different services or different endpoints at a common service.

The following example illustrates using this optional element to specify an endpoint address for the pseudonym service of the federating organization.

```
<fed:PseudonymServiceEndpoint>
  <wsa:Address> http://fabrkam.com/federation/Pseudo </wsa:Address>
</fed:PseudonymServiceEndpoint>
```

3.1.8 AttributeServiceEndpoint Element

The optional `<fed:AttributeServiceEndpoint>` element allows a federation metadata provider to specify the endpoint address of its attribute service which can be referenced by federated partners when requesting tokens from it. This element populates the [Federation Metadata] property. This is typically specified by requestors and is a service-level statement.

The schema for this optional element is shown below.

```
<fed:AttributeServiceEndpoint>
  wsa:EndpointReferenceType
</fed:AttributeServiceEndpoint>
```

The content of this element is an endpoint reference as defined by [WS-Addressing] providing a transport address for the issuer STS. The endpoint reference MAY (and SHOULD if there is no expectation that the policy is known *a priori*) include metadata for the STS endpoint or a reference to an endpoint from where such metadata can be retrieved by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for additional details).

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

It should be noted that this element MAY occur multiple times indicating different services or different endpoints at a common service.

The following example illustrates using this optional element to specify an endpoint address for the attribute service of the federating organization.

```
<fed:AttributeServiceEndpoint>
  <wsa:Address> http://fabrkam.com/federation/Attr </wsa:Address>
</fed:AttributeServiceEndpoint>
```

3.1.9 SingleSignOutSubscripionEndpoint Element

The optional `<fed:SingleSignOutSubscriptionEndpoint>` element allows a federation metadata provider to specify the endpoint address of its subscription service which can be used to subscribe to federated sign-out messages. This element populates the [Federation

Metadata] property. This is typically specified by token issuers and security token services. This is typically a service-level statement but can be an endpoint-level statement.

The schema for this optional element is shown below.

```
<fed:SingleSignOutSubscriptionEndpoint>
  wsa:EndpointReferenceType
</fed:SingleSignOutSubscriptionEndpoint>
```

The content of this element is an endpoint reference as defined by [WS-Addressing] providing a transport address for the subscription manager.

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

3.1.10 SingleSignOutNotificationEndpoint Element

Services MAY subscribe for sign-out notifications however clients MAY also push notifications to services. The optional `<fed:SingleSignOutNotificationEndpoint>` element allows a federation metadata provider to specify the endpoint address to which push notifications of sign-out are to be sent. This element populates the [Federation Metadata] property. This is typically specified by service providers and security token services. This is typically a service-level statement but can be an endpoint-level statement.

The schema for this optional element is shown below.

```
<fed:SingleSignOutNotificationEndpoint>
  wsa:EndpointReferenceType
</fed:SingleSignOutNotificationEndpoint>
```

The content of this element is an endpoint reference as defined by [WS-Addressing] providing a transport address for the subscription manager.

This element allows attributes to be added so long as they do not alter the semantics defined in this specification.

3.1.11 TokenTypesOffered Element

The optional `<fed:TokenTypesOffered>` element allows a federation metadata provider to specify the list of offered security token types that can be issued by its STS. A federated partner can use the offered token types to decide what token type to ask for when requesting tokens from it. This element populates the [Federation Metadata] property. This is typically specified by token issuers and security token services. This is typically a service-level statement but can be an endpoint-level statement.

The schema for this optional element is shown below.

```
<fed:TokenTypesOffered ...>
  <fed:TokenType Uri="xs:anyURI" ...>
    ...
  </fed:TokenType> +
  ...
</fed:TokenTypesOffered>
```

The following describes the elements listed in the schema outlined above:

/fed:TokenTypesOffered

This element is used to express the list of token types that the federating STS is capable of issuing.

/fed:TokenTypesOffered/fed:TokenType

This element indicates an individual token type that the STS can issue.

/fed:TokenTypesOffered/fed:TokenType/@Uri
This attribute provides the unique identifier (URI) of the individual token type that the STS can issue.

/fed:TokenTypesOffered/fed:TokenType/{any}
The semantics of any content for this element are undefined. Any extensibility or use of sub-elements MUST NOT alter the semantics defined in this specification.

/fed:TokenTypesOffered/fed:TokenType/@{any}
This extensibility mechanism allows attributes to be added so long as they don't violate or alter the semantics defined in this specification.

/fed:TokenTypesOffered/@{any}
This extensibility mechanism allows attributes to be added so long as they don't violate or alter the semantics defined in this specification.

/fed:TokenTypesOffered/{any}
The semantics of any content for this element are undefined. Any extensibility or use of sub-elements MUST NOT alter the semantics defined in this specification.

The following example illustrates using this optional element to specify that the issuing STS of the federating organization can issue both SAML 1.1 and SAML 2.0 tokens [WSS:SAMLTokenProfile].

```
<fed:TokenTypesOffered>
  <fed:TokenType Uri="urn:oasis:names:tc:SAML:1.1" />
  <fed:TokenType Uri="urn:oasis:names:tc:SAML:2.0" />
</fed:TokenTypesOffered>
```

3.1.12 UriNamedClaimTypesOffered Element

The optional `<fed:UriNamedClaimTypesOffered>` element allows a federation metadata provider to specify the list of publicly offered claim types, named using URIs, that can be asserted in security tokens issued by its STS. Note that issuers MAY support additional claims and that not all claims may be available for all token types. If other means of describing/identifying claims are used in the future, then corresponding XML elements can be introduced to publish the new claim types. A federated partner can use the offered claim types to decide which claims to ask for when requesting tokens from it. This specification places no requirements on the syntax used to describe the claims. This element populates the [Federation Metadata] property. This is typically specified by token issuers and security token services. This is typically a service-level statement but can be an endpoint-level statement.

The schema for this optional element is shown below.

```
<fed:UriNamedClaimTypesOffered ...>
  (<fed:ClaimType Uri="xs:anyURI" ...>
    <fed:DisplayName ...>xs:string</fed:DisplayName> ?
    <fed:Description ...>xs:string</fed:Description> ?
    ...
  </fed:ClaimType>) +
  ...
</fed:UriNamedClaimTypesOffered>
```

The following describes the elements listed in the schema outlined above:

/fed:UriNamedClaimTypesOffered
This element is used to express the list of claim types that the STS is capable of issuing.

/fed:UriNamedClaimTypesOffered/fed:ClaimType
This required element indicates an individual claim type that the STS can assert.

/fed:UriNamedClaimTypesOffered/fed:ClaimType/@Uri
This required attribute provides the unique name or identifier (as a URI) of the individual claim type that the STS can assert.

/fed:UriNamedClaimTypesOffered/fed:ClaimType/fed:DisplayName
This optional element provides a friendly name for this claim type that can be shown in user interfaces.

/fed:UriNamedClaimTypesOffered/fed:ClaimType/fed:DisplayName/@{any}
This extensibility point allows attributes to be added so long as they don't alter the semantics defined in this specification.

/fed:UriNamedClaimTypesOffered/fed:ClaimType/fed:Description
This optional element provides a description of the semantics for this claim type.

/fed:UriNamedClaimTypesOffered/fed:ClaimType/fed:Description/@{any}
This extensibility point allows attributes to be added so long as they don't alter the semantics defined in this specification.

/fed:UriNamedClaimTypesOffered/fed:ClaimType/{any}
This extensibility point allows additional informative elements to be added so long as they don't alter the semantics defined in this specification.

/fed:UriNamedClaimTypesOffered/fed:ClaimType/@{any}
This extensibility point allows attributes to be added so long as they don't alter the semantics defined in this specification.

/fed:UriNamedClaimTypesOffered/@{any}
This extensibility point allows attributes to be added so long as they don't alter the semantics defined in this specification.

The following example illustrates using this optional element to specify that the issuing STS of the federating organization can assert two claim types named using URIs.

```
<fed:UriNamedClaimTypesOffered>
  <fed:ClaimType Uri="http://.../claims/EmailAddr">
    <fed:DisplayName>Email Address</fed:DisplayName>
  </fed:ClaimType>
  <fed:ClaimType Uri="http://.../claims/IsMember">
    <fed:DisplayName>Is a Member (yes/no)</fed:DisplayName>
    <fed:Description>If a person is a member of this club</fed:Description>
  </fed:ClaimType>
</fed:UriNamedClaimTypesOffered>
```

3.1.13 AutomaticPseudonyms Element

The optional `<fed:AutomaticPseudonyms>` element allows a federation metadata provider to indicate if it automatically maps pseudonyms or applies some form of identity mapping. This element populates the [Federation Metadata] property. This is typically specified by token issuers and security token services. This is typically a service-level statement but can be an endpoint-level statement. If not specified, requestors should assume that the service does not perform automatic mapping (although it MAY).

The schema for this optional element is shown below.

```
<fed:AutomaticPseudonyms>
  xs:boolean
</fed:AutomaticPseudonyms>
```

3.1.14 [Signature] Property

The optional [Signature] property provides a digital signature over the federation metadata document to ensure data integrity and provide data origin authentication. The provider of a federation metadata document SHOULD include a digital signature over the metadata document, and consumers of the metadata document SHOULD perform signature verification if a signature is present.

The token used to sign this document MUST speak for the endpoint. If the metadata is for a token issuer then the key used to sign issued tokens SHOULD be used to sign this document. This means that if a `<fed:TokenSigningKey>` is specified, it SHOULD be used to sign this document.

This section describes the use of [XML-Signature] to sign the federation metadata document, but other forms of digital signatures MAY be used for the [Signature] property. XML Signature is the RECOMMENDED signing mechanism. The [Signature] property (in the case of XML Signature this is represented by the `<ds:Signature>` element) provides the ability for a federation metadata provider organization to sign the metadata document such that a partner organization consuming the metadata can authenticate its origin.

The signature over the federation metadata document MUST be signed using an enveloped signature format as defined by the [XML-Signature] specification. In such cases the root of the signature envelope MUST be the `<fed:FederationMetadata>` element as shown in the following example. If the metadata document is included inside another XML document, such as a SOAP message, the root of the signature envelope MUST remain the same. Additionally, XML Exclusive Canonicalization [XML-C14N] MUST be used when signing with [XML-Signature].

```
(01) [<?xml version='1.0' encoding=... > ]
(02) <fed:FederationMetadata
(03)   xmlns:fed="..." xmlns:ds="..."
(04)   wsu:Id="_fedMetadata">
(05)   ...
(06)   <ds:Signature xmlns:ds="...">
(07)     <ds:SignedInfo>
(08)      <ds:CanonicalizationMethod Algorithm="..." />
(09)      <ds:SignatureMethod Algorithm="..." />
(10)      <ds:Reference URI="_fedMetadata">
(11)       <ds:Transforms>
(12)        <ds:Transform Algorithm=".../xmldsig#enveloped-signature" />
(13)        <ds:Transform Algorithm=".../xml-exc-c14n#" />
(14)       </ds:Transforms>
(15)       <ds:DigestMethod Algorithm="..." />
(16)       <ds:DigestValue>xdJRPBPERvaZD9gTt4e6Mg==</ds:DigestValue>
(17)      </ds:Reference>
(18)     </ds:SignedInfo>
(19)     <ds:SignatureValue> mpcFEK6JuUFBPoJQ8VBW2Q==</ds:SignatureValue>
(20)     <ds:KeyInfo>
(21)     ...
(22)     </ds:KeyInfo>
```

```
(23)    </ds:Signature>
(24) </fed:FederationMetadata>
```

Note that the enveloped signature contains a single `ds:Reference` element (line 10) containing a URI reference to the `<fed:FederationMetadata>` root element (line 04) of the metadata document.

3.1.15 Example Federation Metadata Document

The following example illustrates a signed federation metadata document that uses the optional metadata elements described above and an enveloped [XML Signature] to sign the document.

```
<?xml version="1.0" encoding="utf-8" ?>
<fed:FederationMetadata wsu:Id="_fedMetadata"
 xmlns:fed="..."  xmlns:wsu="..." xmlns:wsse="..." xmlns:ds="..."
 xmlns:wsa="...">
 <fed:Federation>
  <fed:TokenSigningKeyInfo>
    <wsse:SecurityTokenReference>
      <ds:X509Data>
        <ds:X509Certificate>
          MIIBsTCCAV+g...zRn3ZVIcvbQE=
        </ds:X509Certificate>
      </ds:X509Data>
    </wsse:SecurityTokenReference>
  </fed:TokenSigningKeyInfo>
  <fed:TokenIssuerName>
    http://fabrikam.com/federation/corporate
  </fed:TokenIssuerName>
  <fed:TokenIssuerEndpoint>
    <wsa:Address> http://fabrkam.com/federation/STS </wsa:Address>
  </fed:TokenIssuerEndpoint>
  <fed:TokenTypesOffered>
    <fed:TokenType Uri="urn:oasis:names:tc:SAML:1.1" />
    <fed:TokenType Uri="urn:oasis:names:tc:SAML:2.0" />
  </fed:TokenTypesOffered>
  <fed:UriNamedClaimTypesOffered>
    <fed:ClaimType Uri="http://.../claims/EmailAddr">
      <fed:DisplayName>Email Address</fed:DisplayName>
    </fed:ClaimType>
    <fed:ClaimType Uri="http://.../claims/IsMember">
      <fed:DisplayName>Is a Member (yes/no)</fed:DisplayName>
      <fed:Description>If a person is a member of this club</fed:Description>
    </fed:ClaimType>
  </fed:UriNamedClaimTypesOffered>
 </fed:Federation>
```

```
 <ds:Signature xmlns:ds="...">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="..." />
      <ds:SignatureMethod Algorithm="..." />
      <ds:Reference URI="_fedMetadata">
        <ds:Transforms>
          <ds:Transform Algorithm=".../xmldsig#enveloped-signature" />
          <ds:Transform Algorithm=".../xml-exc-c14n#" />
        </ds:Transforms>
        <ds:DigestMethod Algorithm="..." />
        <ds:DigestValue>xdJRPBPERvaZD9gTt4e6Mg==</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>mpcFEK6JuUFBPoJQ8VBW2Q==</ds:SignatureValue>
    <ds:KeyInfo>
      ...
    </ds:KeyInfo>
  </ds:Signature>
</fed:FederationMetadata>
```

3.2. Acquiring the Federation Metadata Document

This section provides specific details and restrictions on how a party may securely obtain the federation metadata document for a *target domain* representing a target organization it wishes to federate with. It should be noted that some providers of federation metadata documents MAY require authentication of requestors or MAY provide different (subset) documents if requestors are not authenticated.

It is assumed that the target domain is expressed as a fully-qualified domain name (FQDN). In other words, it is expressed as the DNS domain name of the target organization, *e.g.*, fabrikam.com.

It should be noted that compliant services are NOT REQUIRED to support all of the mechanisms defined in this section. If a client only has a DNS host name and wants to obtain the federation metadata, the following order is the RECOMMENDED bootstrap search order:

1. Use the well-known HTTPS address with the federation ID
2. Use the well-known HTTPS address for the default federation
3. Use the well-known HTTP address with the federation ID
4. Use the well-known HTTP address for the default federation
5. Look for any DNS SRV records indicating federation metadata locations

If multiple locations are available and no additional prioritization is specified, the following order is the RECOMMENDED download processing order:

1. HTTPS
2. WS-Transfer/WS-ResourceTransfer
3. HTTP

3.2.1 WSDL

The metadata document MAY be included within a WSDL document using the extensibility mechanisms of WSDL. Specifically the `<fed:FederationMetadata>` element can be placed inside of WSDL documents in the same manner as policy documents are as specified in WS-PolicyAttachment.

The metadata document can appear in WSDL for a service, port, or binding.

3.2.2 The Federation Metadata Path

A default path MAY be supported to provide federation metadata. The path for obtaining the federation metadata document for the default federation for a target domain denoted by ***target-DNS-domain*** SHOULD be constructed as follows:

http://*server-name*/**FederationMetadata**/*spec-version*/**FederationMetadata.xml**

or

https://*server-name*/**FederationMetadata**/*spec-version*/**FederationMetadata.xml**

where

server-name is the host name (DNS name) of a server providing the federation metadata document. It SHOULD be obtained by doing a DNS query of SRV records for ***target-DNS-domain*** as described in Section 3.2.6. If no DNS record is found, then the target DNS domain name MUST BE used as the default value of the server name as well.

spec-version is the version of the federation metadata specification supported by the acquiring party. For this version of the specification the ***spec-version*** MUST BE the string "**2006-12**".

Implementations MAY choose to use a short form of the target DNS domain name, such as the primary domain and suffix, but this choice is implementation specific.

The following subsections describe the mechanisms through which the federation metadata document for a target domain may be acquired by a federating party. The target domain MUST support at least one of the mechanisms described below, but MAY choose to support more than one mechanism.

It is RECOMMENDED that a target domain (or organization) that makes federation metadata available for acquisition by partners SHOULD publish DNS SRV resource records to allow an acquiring party to locate the servers where the metadata is available. The type and format of the SRV resource records to be published in DNS is described in Section 3.2.6. These records correspond to each metadata acquisition mechanism specified in the following subsections.

If a specific federation context is known, the following URLs SHOULD be checked prior to checking for the default federation context.

http://*server-name*/**FederationMetadata**/*spec-version*/fed-id/**FederationMetadata.xml**

or

https://*server-name*/**FederationMetadata**/*spec-version/fed-id*/**FederationMetadata.xml**

where

fed-id is the `FederationID` value described previously for identifying a specific federation.

3.2.3 Retrieval Mechanisms

The following OPTIONAL retrieval mechanisms are defined:

Using HTTP

The federation metadata document may be obtained from the following URL using HTTP GET mechanism:

`http:`***`path`***

where ***path*** is constructed as described in Section 3.2.2.

Metadata signatures are RECOMMENDED when using HTTP download.

Using HTTPS

The federation metadata document may be obtained from the following URL using HTTPS GET mechanism:

```
https:path
```

where ***path*** is constructed as described in Section 3.2.2.

There is no requirement that the HTTPS server key be related to the signing key identified in the metadata document, but it is RECOMMENDED that requestors verify that both keys can speak for the target service.

Using WS-Transfer/WS-ResourceTransfer

The federation metadata document can be obtained by sending the [WS-Transfer] "Get" operation to an endpoint that serves that metadata as described in [WS-MetadataExchange] (see also section 3.2.5). Note that the [WS-ResourceTransfer] extensions MAY be used to filter the metadata information returned.

The use of [WS-Security] with [WS-Transfer/WS-ResourceTransfer] is RECOMMENDED to authenticate the sender and protect the integrity of the message.

3.2.4 FederatedMetadataHandler Header

If an endpoint reference for metadata obtained via SOAP requests is not already available to a requester (e.g. when only a URL is know), the requestor SHOULD include the `<fed:FederationMetadataHandler>` header to allow metadata requests to be quickly identified. The syntax is as follows:

```
<fed:FederationMetadataHandler .../>
```

The`<fed:FederationMetadataHandler>` header SHOULD NOT use a `S:mustUnderstand='1'` attribute. Inclusion of this header allows a front-end service to know that federation metadata is being requested and perform header-based routing.

The following example illustrates a [WS-Transfer] with [WS-ResourceTransfer] extensions request message to obtain the federation metadata document for an organization with contoso.com as its domain name.

```
(01) <s12:Envelope
(02)     xmlns:s12="..."
(03)     xmlns:wsa="..."
(04)     xmlns:wsxf="..."
(05)     xmlns:fed="...">
(06)   <s12:Header>
(07)     <wsa:Action>
(08)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
(09)     </wsa:Action>
(10)     <wsa:MessageID>
(11)         uuid:73d7edfd-5c3d-b949-46ba-02decaee433f
(12)     </wsa:MessageID>
(13)     <wsa:ReplyTo>
(14)       <wsa:Address>http://fabrikam.com/Endpoint</wsa:Address>
(15)     </wsa:ReplyTo>
(16)     <wsa:To>
```

```
(17)       http://contoso.com/FederationMetadata/2006-12/FederationMetadata.xml
(18)     </wsa:To>
(19)     <fed:FederatedMetadataHandler />
(20)   </s12:Header>
(21)   <s12:Body />
(22) </s12:Envelope>
```

The response to the [WS-Transfer] with [WS-ResourceTransfer] extensions request message is illustrated below.

```
(01) <s12:Envelope
(02)     xmlns:s12="..."
(03)     xmlns:wsa="..."
(04)     xmlns:wsxf="..."
(05)     xmlns:fed="...">
(06)   <s12:Header>
(07)     <wsa:To>http://fabrikam.com/Endpoint</wsa:To>
(08)     <wsa:Action>
(09)       http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
(10)     </wsa:Action>
(11)     <wsa:MessageID>
(12)       uuid:86d7eac5-6e3d-b869-64bc-35edacee743d
(13)     </wsa:MessageID>
(14)     <wsa:RelatesTo>
(15)       uuid:73d7edfd-5c3d-b949-46ba-02decaee433f
(16)     </wsa:RelatesTo>
(17)   </s12:Header>
(18)   <s12:Body>
(19)     <fed:FederationMetadata
(20)         xmlns:fed="...">
(21)       ...
(22)     </fed:FederationMetadata>
(21)  </s12:Body>
(22) </s12:Envelope>
```

3.2.5 Metadata Exchange Dialect

The federation metadata document can be included as a metadata unit within a Web service `<mex:Metadata>` element, which is a collection of metadata units, using the metadata unit inclusion mechanisms described in [WS-MetadataExchange]. This can be done by including a `<mex:MetadataSection>` element that contains the federation metadata document in-line or by reference. To facilitate inclusion of the federation metadata as a particular type of metadata unit, the following metadata dialect URI is defined in this specification that MUST be used as the value of the `<mex:MetadataSection/@Dialect>` XML attribute:

```
        http://schemas.xmlsoap.org/ws/2006/12/federation
```

No identifiers for federation metadata units, as specified by the value of the optional `<mex:MetadataSection/@Identifier>` XML attribute, are defined in this specification.

For example, a federation metadata unit specified in-line within a `<mex:Metadata>` element is shown below:

```
<mex:Metadata>
  <mex:MetadataSection
           Dialect='http://schemas.xmlsoap.org/ws/2006/12/federation'>
    <fed:FederationMetadata ...>
      ...
    </fed:FederationMetadata>
  <mex:MetadataSection>
<mex:Metadata>
```

3.2.6 Publishing Federation Metadata Location

A target domain (or organization) that makes federation metadata available for acquisition by partners should publish SRV resource records in the DNS database to allow an acquiring party to locate the servers where the metadata is available. The specific format and content of the SRV resource records to be published is described here.

The SRV record is used to map the name of a service (in this case the federation metadata service) to the DNS hostname of a server that offers the service. For more information about SRV resource records, see [DNS-SRV-RR]. The general form of the *owner name* of a SRV record to be published is as follows:

```
_Service.Protocol.TargetDnsDomain
```

In this case, a target domain offers the "federation metadata" service over one or more of the protocol mechanisms described earlier (namely, HTTP, HTTPS or WS-Transfer/WS-ResourceTransfer). For each protocol mechanism supported by a target domain, a corresponding SRV record should be published in DNS as follows.

If acquisition of the federation metadata document using HTTP GET (Section 3.2.3) is supported, then the owner name of the published SRV record must be of the form below:

```
_fedMetadata._http.TargetDnsDomain
```

If acquisition of the federation metadata document using HTTPS GET (Section 3.2.3) is supported, then the owner name of the published SRV record must be of the form below:

```
_fedMetadata._https.TargetDnsDomain
```

If acquisition of the federation metadata document using [WS-Transfer/WS-ResourceTransfer] (Section 3.2.3) is supported, then the owner name of the published SRV record must be of the form below:

```
_fedMetadata._wsxfr._http.TargetDnsDomain
```

The remaining information included in the SRV record content is as follows:

Priority The priority of the server. Clients attempt to contact the server with the lowest priority and move to higher values if servers are unavailable (or not desired).

Weight A load-balancing mechanism that is used when selecting a target server from those that have the same priority. Clients can randomly choose a server with probability proportional to the weight.

Port The port where the server is listening for the service.

Target The fully-qualified domain name of the host server.

Note that if multiple protocols are specified with the same priority, the requestor MAY use any protocol or process in any order it chooses.

The following example illustrates the complete SRV records published by the organization with domain name "contoso.com" that makes its federation metadata available over all three mechanisms discussed earlier.

```
server1.contoso.com  IN  A  128.128.128.0
server2.contoso.com  IN  A  128.128.128.1
_fedMetadata._http.contoso.com  IN  SRV 0 0 80 server1.contoso.com
_fedMetadata._https.contoso.com  IN  SRV 0 0 443 server1.contoso.com
_fedMetadata._wsxfr.contoso.com  IN  SRV 0 0 80 server2.contoso.com
```

A client attempting to acquire the federation metadata for a target domain using any selected protocol mechanism should query DNS for SRV records using one of the appropriate owner name forms described above.

3.2.7 Federation Metadata Acquisition Security

It is RECOMMENDED that a target domain publishing federation metadata SHOULD include a signature in the metadata document using a key that is authorized to "speak for" the target domain. If the metadata contains a `<fed:TokenSigningKey>` element then this key SHOULD be used for the signature. If there are multiple `Federation` elements specified then the default scope's signing key SHOULD be used. If there is no default scope then the choice is up to the signer. Recipients of federation metadata SHOULD validate that signature to authenticate the metadata publisher and verify the integrity of the data. Specifically, a recipient SHOULD verify that the key used to sign the document has the right to "speak for" the target domain (see *target-DNS-domain* in Section 3.2.2) with which the recipient is trying to federate. See also the security considerations at the end of this document.

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, but which are no longer required. In typical usage, sign-out notification serves as a hint – upon termination of a principal's session – that it is OK to flush cached data (such as security tokens) or state information for that specific principal. It should be noted that a sign-out message is a *one-way* message (that is, it doesn't have a reply). Depending on the type of application that is being used to sign-out, the process may vary. For example, the implication of sign-out on currently active transactions is undefined and is resource-specific.

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.

Sign-out is separate from token cancellation as it applies to all tokens and all target sites for the principal within the domain/realm.

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 defines optional sign-out messages that MAY be used. It should be noted, however, that the typical usage pattern is that only token issuance and message security are used and sign-out messages are only for special scenarios. Sign-out messages, whether from the client to the IP/STS, from the IP/STS to a subscriber, or from the client to a service provider, all use the same message form described in this section.

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

```
http://schemas.xmlsoap.org/2006/12/Federation/SignOut
```

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

```
<fed:SignOut wsu:Id="..." ...>
    <fed:Realm>xs:anyURI</fed:Realm> ?
    <fed:SignOutBasis ...>...<fed:SignOutBasis>
    ...
</fed:SignOut>
```

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

/fed:SignOut
: This element represents a sign-out message.

/fed:SignOut/fed:Realm
: This optional element specifies the "realm" to which the sign-out applies and is specified as a URI. If no realm is specified, then it is assumed that the recipient understands and uses a fixed/default realm.

/fed:SignOut/fed:SignOutBasis
: The contents of this REQUIRED element 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.

/fed:SignOut/fed:SignOutBasis/@{any}
: This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element so long as they don't alter the semantics of this specification

/fed:SignOut/fed:SignOutBasis/{any}
: This is an extensibility mechanism to allow the inclusion of the relevant security token reference or security token(s).

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

/fed:SignOut/@{any}
: This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element so long as they don't alter the semantics of this specification.

/fed:SignOut/{any}
: This is an extensibility mechanism to allow additional elements to be used. For example, an STS might use extensibility to further qualify the sign-out basis.

The <fed:SignOut> message SHOULD be signed by the requestor 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 (see WS-Security for further discussion on this). 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 <fed:SignOut> message:

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
      xmlns:wsu="..." xmlns:wsse="...">
    <S:Header>
        ...
        <wsu:Timestamp wsu:Id="ts">
            ...
        </wsu:Timestamp>
        <wsse:Security>
            <!-- Signature referecing IDs "ts" & "so" -->
            ...
        </wsse:Security>
    </S:Header>
    <S:Body>
        <fed:SignOut wsu:Id="so">
          <fed:SignOutBasis>
            <wsse:UsernameToken>
                <wsse:Username>NNK</wsse:Username>
            </wsse:UsernameToken>
          </fed:SignOutBasis>
        </fed: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 typical use case, federated sign-out messages will be generated by the principal terminating a session, either at the "primary STS" (the IP/STS that manages the principal's identity) or at one of the resource providers (or its STS) accessed during the session. There are two primary flows for these messages. In one case they are effectively chained through all the STSs involved in the session; that is, a mechanism is used (if available) by the "primary STS" to send sign-out messages to all the other STSs in a sequential manner by causing each message to cause the next message to occur in sequence resulting in a message back to itself either on completion or at each step to orchestrate the process. The second approach is to require the "primary STS" to send sign-out messages to all the other token services and target services in parallel (those that it knows about).

The chained (sequential) approach has been found to be fragile. If one of the message fails to complete its local processing and does not pass the sign-out message on – or the network partitions – the sign-out notification does not reach all the involved parties. For this reason, compliant implementations SHOULD employ the parallel approach. If the session is terminated at a resource provider, it SHOULD clean up any local state and then send a sign-out message to the "primary STS". The latter SHOULD send parallel sign-out messages to all the other STSs.

Sessions may involve secondary branches (between token services at different resources) of which the "primary STS" has no knowledge. In these cases, the appropriate resource token services SHOULD perform the role of "primary STS" for sign-out of these branches.

It should be noted that clients MAY also push (send) sign-out messages directly to other services such as secondary IP/STSs or service providers.

Sign-out could potentially be applied to one of two different scopes for the principal's session. Sign-out initiated at the "primary STS" SHOULD have global scope and apply to all resource STSs and all branches of the session. Sign-out initiated at a resource STS could also have global scope as described above. However, it could also be considered as a request to clean up only the session state related to that particular resource provider. Thus implementations MAY provide a mechanism to restrict the scope of federated sign-out requests that originate at a resource STS to its particular branch of the principal's session. This SHOULD result in cleaning up all state at (or centered upon) that STS. It SHOULD involve a request to be sent to the "primary STS" to clean up session state only for that particular STS or resource provider.

Federated sign-out request processing could involve providing status messages to the user. This behavior is implementation specific and out-of-scope of this specification.

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.

SSO messages are obtained by subscribing to the subscription endpoint using the mechanisms described in [WS-Eventing]. The subscription endpoint, if available, is described in the federation metadata document.

The [WS-Eventing] mechanisms allow for subscriptions to be created, renewed, and cancelled. SSO subscriptions can be filtered using the XPath filter defined in [WS-Eventing] or using the SSO filter specified by the following URI:

```
http://schemas.xmlsoap.org/ws/2006/12/federation/ssoevt
```

This filter allows the specification of a realm and security tokens to restrict the SSO messages. The syntax is as follows:

```
<wse:Subscribe ...>
  ...
  <wse:Filter Dialect=".../federation/ssoevt">
    <fed:Realm>...</fed:Realm> ?
    ...security tokens...
  </wse:Filter>
  ...
</wse:Subscribe>
```

The following describes elements and attributes illustrated above:

/wse:Filter/fed:Realm

This optional element specifies the "realm" to which the sign-out applies. At most one `<fed:Realm>` can be specified. The contents of this element are the same type and usage as in the `fed:Signout/fed:Realm` described above. If this element is not specified it is assumed that either the subscription service knows how to infer the correct realm and uses a single service-determined realm or the request fails. Note that if multiple realms are desired then multiple subscriptions are needed.

/wse:Filter/… 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 `<wsse:UsernameToken>` will be the most common. Note that if multiple tokens are specified they represent a logical OR – that is, messages that match any of the tokens for the corresponding realm are reported.

This filter dialect does not allow any contents other than those described above. If no filter is specified then the subscription service MAY fail or MAY choose a default filter for the subscription.

5. Attribute Service

Web services often need to be able to obtain additional data related to service requestors to provide the requestor with a richer (e.g. personalized) experience. This can be addressed by having an attribute service that requesters and services MAY use to access this additional information. In many cases, the release of this information about a service requestor is subject to authorization and privacy rules and access to this data (or the separate service that has data available for such purposes) is only granted to authorized services for any given attribute.

Attribute stores most likely exist in some form already in service environments using service-specific protocols (e.g. such as LDAP). An attribute service provides the interface to this attribute store.

Figure 21 below illustrates the conceptual namespace of an 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 if a principal exists, any corresponding attribute data should 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.

Principals and resources may have specific policies that are required when accessing and managing their attributes. Such policies use the [WS-Policy] framework. As well, these principals (and resources) MAY be specified as domain expressions to scope policy assertions as described in [WS-PolicyAttachment].

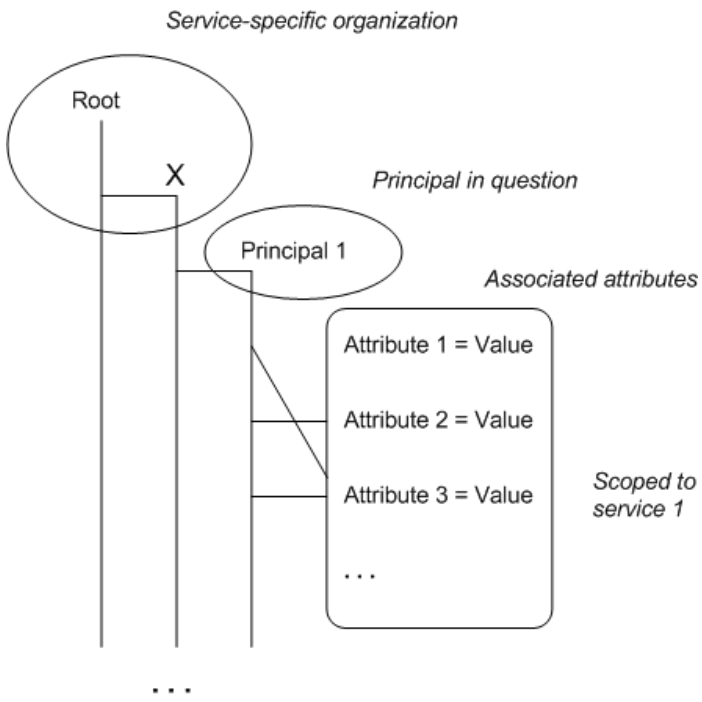


Figure 21 Attribute Service

It is expected that separate attributes MAY be shared differently and MAY 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 by definition of new policy assertions indicating the attribute service supported.

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

This specification does not require or define a specific attribute service definition or interface.

6. Pseudonym Service

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

Pseudonym services may exist in some form already in service environments using service-specific 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:

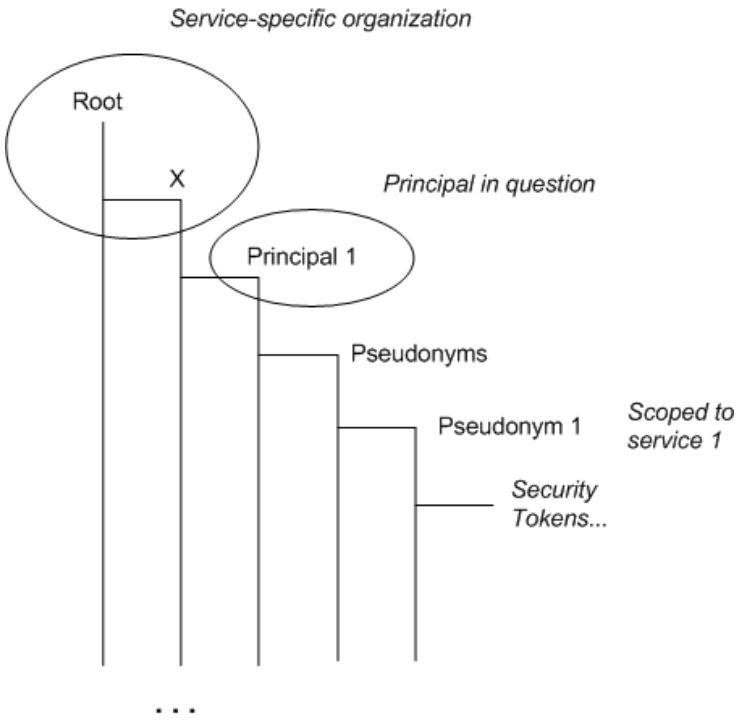


Figure 22 Pseudonym Service

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 pseudonyms may have different access control and privacy policies and that these can vary by principal or by scope within principal. Consequently, each pseudonym 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).

Pseudonym services MUST support the interfaces defined in this section for getting, setting, and deleting pseudonyms.

6.1. Filtering Pseudonyms

When performing operations on a pseudonym store it is RECOMMENDED to filter the scope of the operation. This is done using the following dialect with the [WS-ResourceTransfer] extensions to [WS-Transfer]:

```
http://schemas.xmlsoap.org/ws/2006/12/federation/pseudonymdialect
```

Alternatively, the <fed:FilterPseudonyms> header MAY be specified with WS-Transfer to allow filtering to be specified as part of an endpoint reference (EPR).

The syntax for the <fed:FilterPseudonyms> element is as follows:

```
<fed:FilterPseudonyms ...>
    <fed:PseudonymBasis ...>...</fed:PseudonymBasis> ?
    <fed:RelativeTo ...>...</fed:RelativeTo> ?
    ...
</fed:FilterPseudonyms>
```

The following describes elements and attributes used in a <fed:FilterPseudonyms> element.

/fed:FilterPseudonyms
: This element indicates a request to filter a pseudonym operation based on given identity information and applicability scope.

/fed:FilterPseudonyms/fed:PseudonymBasis
: This element specifies a security token or security token reference identifying the known identity information. This element is typically required to identify the basis but may be omitted if the context is known. This specification places no requirements on what information (claims) are required to be a pseudonym basis – that can vary by service.

/fed:FilterPseudonyms/fed:PseudonymBasis/@{any}
: This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:FilterPseudonyms/fed:PseudonymBasis/{any}
: This is an extensibility mechanism to allow the inclusion of the relevant security token reference or security token.

/fed:FilterPseudonyms/fed:RelativeTo
: This RECOMMENDED element indicates the scope for which the pseudonym is requested. This element has the same type as <wsp:AppliesTo>.

/fed:FilterPseudonyms/fed:RelativeTo/@{any}
: This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:FilterPseudonyms/@{any}
: This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:FilterPseudonyms/{any}
: This is an extensibility point allowing content elements to be specified. Note that no element can alter semantic defined in this specification.

As noted above, in some circumstances it MAY be desirable to include a filter as part of an EPR. To accommodate this, <fed:FilterPseudonyms> element MAY be specified as a SOAP header. It is RECOMMENDED that the SOAP *mustUnderstand* attribute be specified as *true* whenever this is used as a header. If a <fed:FilterPseudonyms> header is specified, the message MUST NOT contain additional filtering.

6.2. Getting Pseudonyms

Pseudonyms are requested from a pseudonym service using the [WS-Transfer] "GET" method with the [WS-ResourceTransfer] extensions. The dialect defined in 6.1 (or the `<fed:FilterPseudonyms>` header) is used to restrict the pseudonyms that are returned.

Pseudonyms are returned in the body of the GET response message in a `<fed:Pseudonym>` element as follows:

```
<fed:Pseudonym ...>
  <fed:PseudonymBasis ...>...</fed:PseudonymBasis>
  <fed:RelativeTo ...>...</fed:RelativeTo>
  <wsu:Expires>...</wsu:Expires>
  <fed:SecurityToken ...>...</fed:SecurityToken> *
  <fed:ProofToken ...>...</fed:ProofToken> *
  ...
</fed:Pseudonym>
```

The following describes elements and attributes described above:

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

/fed:Pseudonym/fed:PseudonymBasis
: This element specifies a security token or security token reference identifying the known identity information (see [WS-Security]). Often this is equivalent to the basis in the request although if multiple pseudonyms are returned that value may be different.

/fed:Pseudonym/fed:PseudonymBasis/@{any}
: This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:Pseudonym/fed:PseudonymBasis/{any}
: This is an extensibility mechanism to allow the inclusion of the relevant security token reference or security token.

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

/fed:Pseudonym/fed:RelativeTo/@{any}
: This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:Pseudonym/wsu:Expires
: This optional element indicates the expiration of the pseudonym.

/fed:Pseudonym/fed:SecurityToken
: This optional element indicates a security token for the scope. Note that multiple tokens MAY be specified.

/fed:Pseudonym/fed:SecurityToken/@{any}
: This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:Pseudonym/fed:SecurityToken/{any}
: This is an extensibility mechanism to allow the inclusion of the relevant security token(s).

/fed:Pseudonym/fed:ProofToken
This optional element indicates a proof token for the scope. Note that multiple tokens MAY be specified.

/fed:Pseudonym/fed:ProofToken/@{any}
This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:Pseudonym/fed:ProofToken/{any}
This is an extensibility mechanism to allow the inclusion of the relevant security token(s).

/fed:Pseudonym/@{any}
This is an extensibility point allowing attributes to be specified. Note that no attribute can alter semantic defined in this specification.

/fed:Pseudonym/{any}
This is an extensibility point allowing content elements to be specified. Note that no element can alter semantic defined in this specification.

For example, the following example obtains the local pseudonym associated with the identity (indicated binary security token) for the locality (target scope) indicated by the URI http://www.fabrikam123.com/NNK.

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
      xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
  <S:Body>
    <wsrt:Get
 Dialect="http://schemas.xmlsoap.org/ws/2006/12/federation/pseudonymdialect">
     <wsrt:Expression>
      <fed:FilterPseudonyms>
          <fed:PseudonymBasis>
              <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
          </fed:PseudonymBasis>
          <fed:RelativeTo>
            <wsa:Address>
                  http://www.fabrikam123.com/NNK
            </wsa:Address>
          </fed:RelativeTo>
      </fed:FilterPseudonyms>
     </wsrt:Expression>
    </wsrt:Get>
  </S:Body>
</S:Envelope>
```

A sample response might be as follows:

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
      xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
```

```
  <S:Body>
   <wsrt:GetResponse>
    <wsrt:Result>
        <fed:Pseudonym>
          <fed:RelativeTo>
            <wsa:Address>
              http://www.fabrikam123.com/NNK
            </wsa:Address>
          </fed:RelativeTo>
          <wsu:Expires>2003-12-10T09:00Z</wsu:Expires>
          <fed:SecurityToken>...</fed:SecurityToken>
          <fed:ProofToken>...</fed:ProofToken>
        </fed:Pseudonym>
    </wsrt:Result>
   </wsrt:GetResponse>
  </S:Body>
</S:Envelope>
```

6.3. Setting Pseudonyms

Pseudonyms are updated in a pseudonym service using the [WS-Transfer] "PUT" operation with the [WS-ResourceTransfer] extensions using the dialect defined in 6.1 (or the `<fed:FilterPseudonyms>` header). This allows one or more pseudonyms to be added. If a filter is not specified, then the PUT impacts the full pseudonym set. It is RECOMMENDED that filters be used.

The following example sets pseudonym associated with the identity (indicated binary security token) for the locality (target scope) indicated by the URI http://www.fabrikam123.com/NNK.

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
      xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
  <S:Body>
   <wsrt:Put
 Dialect="http://schemas.xmlsoap.org/ws/2006/12/federation/pseudonymdialect">
    <wsrt:Fragment Mode="Inset">
     <wsrt:Expression>
      <fed:FilterPseudonyms>
          <fed:PseudonymBasis>
              <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
          </fed:PseudonymBasis>
```

```
          <fed:RelativeTo>
            <wsa:Address>
                  http://www.fabrikam123.com/NNK
            </wsa:Address>
          </fed:RelativeTo>
      </fed:FilterPseudonyms>
     </wsrt:Expression>
     <wsrt:Value>
        <fed:Pseudonym>
          <fed:PseudonymBasis>
            <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
          </fed:PseudonymBasis>
          <fed:RelativeTo>
            <wsa:Address>
                http://www.fabrikam123.com/NNK
            </wsa:Address>
          </fed:RelativeTo>
          <fed:SecurityToken>
              <wsse:UsernameToken>
                  <wsse:Username> "Nick" </wsse:Username>
              </wsse:UsernameToken>
          </fed:SecurityToken>
          <fed:ProofToken>...</fed:ProofToken>
        </fed:Pseudonym>
      </wsrt:Value>
    </wsrt:Fragment>
    </wsrt:Put>
  </S:Body>
</S:Envelope>
```

6.4. Deleting Pseudonyms

Pseudonyms are deleted in a pseudonym service using the [WS-Transfer] "PUT" operation with the [WS-ResourceTransfer] extensions. The dialect defined in 6.1 (or the `<fed:FilterPseudonyms>` header) is used to restrict the scope of the "PUT" to only remove pseudonym information corresponding to the filter. If a filter is not specified, then the PUT impacts the full pseudonym set. It is RECOMMENDED that filters be used.

The following example deletes the pseudonym associated with the identity (indicated binary security token) for the locality (target scope) indicated by the URI http://www.fabrikam123.com/NNK.

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
      xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
  <S:Body>
   <wsrt:Put
 Dialect="http://schemas.xmlsoap.org/ws/2006/12/federation/pseudonymdialect">
    <wsrt:Fragment Mode="Remove">
     <wsrt:Expression>
      <fed:FilterPseudonyms>
          <fed:PseudonymBasis>
              <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
          </fed:PseudonymBasis>
          <fed:RelativeTo>
            <wsa:Address>
                  http://www.fabrikam123.com/NNK
            </wsa:Address>
          </fed:RelativeTo>
      </fed:FilterPseudonyms>
     </wsrt:Expression>
    </wsrt:Fragment>
   </wsrt:Put>
  </S:Body>
</S:Envelope>
```

6.5. Creating Pseudonyms

Pseudonyms are created in a pseudonym service using the WS-Resource "CREATE" operation with the [WS-ResourceTransfer] extensions. This allows one or more pseudonyms to be added. The dialect defined in 6.1 (or the `<fed:FilterPseudonyms>` header) is specified on the CREATE to only create pseudonym information corresponding to the filter. If a filter is not specified, then the CREATE impacts the full pseudonym set. It is RECOMMENDED that filters be used.

The following example creates pseudonym associated with the identity (indicated binary security token) for the locality (target scope) indicated by the URI http://www.fabrikam123.com/NNK.

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
      xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
```

```
 <S:Body>
  <wsrt:Create
Dialect="http://schemas.xmlsoap.org/ws/2006/12/federation/pseudonymdialect">
   <wsrt:Fragment>
    <wsrt:Expression>
     <fed:FilterPseudonyms>
         <fed:PseudonymBasis>
             <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
         </fed:PseudonymBasis>
         <fed:RelativeTo>
           <wsa:Address>
                 http://www.fabrikam123.com/NNK
           </wsa:Address>
         </fed:RelativeTo>
     </fed:FilterPseudonyms>
    </wsrt:Expression>
    <wsrt:Value>
       <fed:Pseudonym>
         <fed:PseudonymBasis>
           <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
         </fed:PseudonymBasis>
         <fed:RelativeTo>
           <wsa:Address>
               http://www.fabrikam123.com/NNK
           </wsa:Address>
         </fed:RelativeTo>
         <fed:SecurityToken>
             <wsse:UsernameToken>
                 <wsse:Username> "Nick" </wsse:Userename>
             </wsse:UsernameToken>
         </fed:SecurityToken>
         <fed:ProofToken>...</fed:ProofToken>
       </fed:Pseudonym>
    </wsrt:Value>
   </wsrt:Fragment>
  </wsrt:Create>
 </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.

Figure 23 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 (1) and then saves them in the pseudonym service (2). 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 (3). 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).

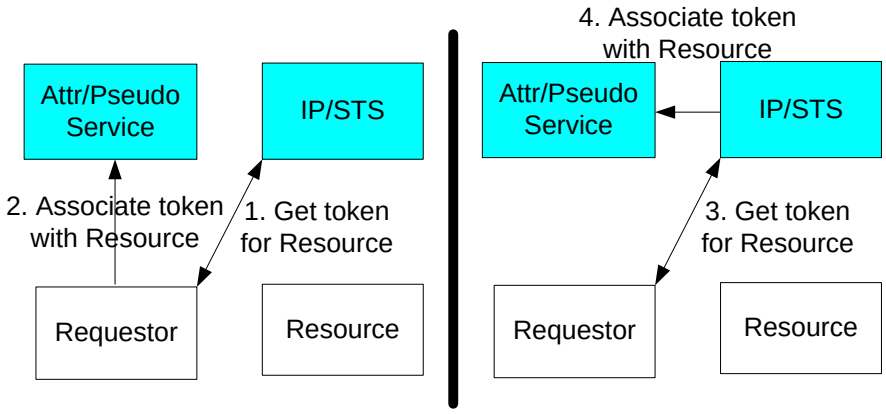


Figure 23: Attribute & Pseudonym Services Relationships to IP/STS Services

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:

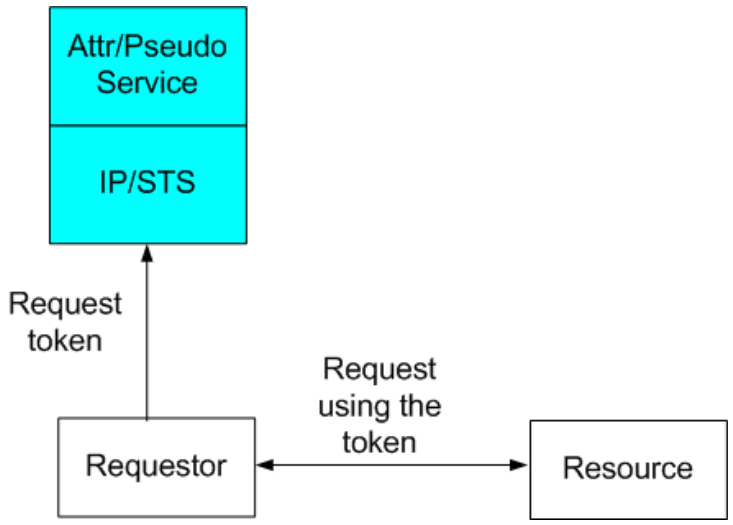


Figure 24: Attribute & Pseudonym Service Fronted by IP/STS

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 discarded by the service.

This approach is an alternative to having the pseudonym service directly return the security token issuance. Both approaches SHOULD be supported 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, an IP/STS MAY automatically map pseudonyms based on the target service. If it doesn't, the following additional options MAY be included in the security token requests using the `<wst:RequestSecurityToken>` request to explicitly request a mapping or to clarify the type of mapping desired.

The following syntax illustrates the RST extension to support these new options:

```
<fed:RequestPseudonym SingleUse="xs:boolean" ? Lookup="xs:boolean" ? ...>
   ...
</fed:RequestPseudonym>
```

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

/fed: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).

/fed: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).

/fed:RequestPseudonym/{any}

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

/fed:RequestPseudonym/@{any}

This is an extensibility mechanism to allow additional attributes to be specified.

If the <RequestPseudonym> isn't present, pseudonym usage/lookup and single use is at the discretion of the IP/STS. Note that if present, as with all RST parameters, processing is at the discretion of the STS and it MAY choose to use its own policy instead of honoring the requestor's parameters.

Note that the above may be echoed in a RSTR response confirming the value used by the STS.

7.2. Usernames and Passwords

If an 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 an 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 subsequently 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. Additional WS-Trust Extensions

The following sub-sections define additional extensions to [WS-Trust] to facilitate federation.

8.1. Reference Tokens

Tokens are exchanged using the mechanisms described in [WS-Trust]. In some cases, however, it is more efficient to not return the token, but return a handle to the token along with the proof information. Requestors can then send messages to services secured with the proof token but only passing the token reference. The recipient is then responsible for obtaining the actual token.

To support this scenario, a reference token MAY be returned in a RSTR response message instead of the actual token. This is a security token and can be used in any way a security token is used; it is just that its contents need to be fetched before they can be processed. Specifically, this token can then be used with [WS-Security] (referenced by ID only) to associate a token with the message. Note that the proof key corresponding to the token

referenced is used to sign messages. The actual token can later be obtained from the issuing party (or its delegate) using the reference provided.

The following URI is defined to identify a reference token within [WS-Security]:

```
http://schemas.xmlsoap.org/ws/2006/12/federation/reftoken
```

The following syntax defines a reference token that can be used in compliance with this specification:

```
<fed:ReferenceToken ...>
  <fed:ReferenceEPR>wsa:EndpointReferenceType</fed:ReferenceEPR> +
  <fed:ReferenceDigest ...>xs:base64Binary</fed:ReferenceDigest> ?
  <fed:ReferenceType ...>xs:anyURI</fed:ReferenceType> ?
  <fed:SerialNo ...>...</fed:SerialNo> ?
  ...
</fed:ReferenceToken>
```

/fed:ReferenceToken
: This specifies a reference token indicating the EPR to which a [WS-Transfer] (optionally with [WS-ResourceTransfer] extensions) GET request can be made to obtain the token.

/fed:ReferenceToken/fed:ReferenceEPR
: The actual EPR to which the [WS-Transfer/WS-ResourceTransfer] GET request is directed. At least one EPR MUST be specified.

/fed:ReferenceToken/fed:ReferenceDigest
: An optional SHA1 digest of token to be returned. The value is the base64 encoding of the SHA1 digest. If the returned token is a binary token, the SHA1 is computed over the raw octets. If the returned token is XML, the SHA1 is computed over the Exclusive XML Canonicalized [XML-C14N] form of the token.

/fed:ReferenceToken/fed:ReferenceDigest/@{any}
: This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:ReferenceToken/fed:ReferenceType
: An optional URI value that indicates the type of token that is being referenced. It is RECOMMENDED that this be provided to allow processors to determine acceptance without having to fetch the token, but in some circumstances this is difficult so it is not required.

/fed:ReferenceToken/fed:ReferenceType/@{any}
: This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:ReferenceToken/fed:SerialNo
: An optional URI value that uniquely identifies the reference token.

/fed:ReferenceToken/fed:SerialNo/@{any}
: This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:ReferenceToken/{any}
: This extensibility mechanism allows additional informative elements to be specified so long as they do not alter the semantics defined in this specification.

/fed:ReferenceToken/@{any}

This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

There are no requirements on the security associated with the handle or dereferencing it. If the resulting token is secured or does not contain sensitive information the STS MAY just make it openly accessible. Alternatively, the STS MAY use the `<wsp:AppliesTo>` information from the RST to secure the token such that only requestors that can speak for that address can obtain the token.

8.2. Indicating Federations

In some scenarios an STS, resource provider, or service provider may be part of multiple federations and allow token requests at a single endpoint that could be processed in the context of any of the federations (so long as the requestor is authorized). In such cases, there may be a need for the requestor to identify the federation context in which it would like the token request to be processed.

The following `<fed:FederationID>` element can be included in a RST (as well as an RSTR):

```
<fed:FederationID ...>xs:anyURI</fed:FederationID>
```

/fed:FederationID

This element identifies the federation context as a URI value in which the token request is made (or was processed).

/fed:FederationID/@{any}

This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

Note that if a `FederationID` is not specified, the *default* federation is assumed.

8.3. Obtaining Proof Tokens from Validation

A requestor may obtain a token for a federation for which the recipient service doesn't actually have the rights to use and extract the session key. For example, when a requestor's IP/STS and the recipient's IP/STS have an arrangement and share keys but the requestor and recipient only describe federation between themselves. In such cases, the requestor and the recipient must obtain the session keys (proof tokens) from their respective IP/STS. For the requestor this is returned in the proof token of its request.

For the recipient, it must pass the message to its IP/STS to have it validated. As part of the validation process, the proof token can be requested by including the parameter below in the RST. When this element is received by an IP/STS, it indicates a desire to have a `<wst:RequestedProofToken>` returned with the session key so that the recipient does not have to submit subsequent messages for validation.

The syntax of the `<fed:RequestProofToken>` is as follows:

```
<fed:RequestProofToken ...>
  ...
</fed:RequestProofToken>
```

/fed:RequestProofToken

When used with a *Validate* request this indicates that the requestor would like the STS to return a proof token so that subsequent messages using the same token/key can be processed by the recipient directly.

/fed:RequestProofToken/@{any}

This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:RequestProofToken/{any}

This contents of this element are undefined and MAY be extended so long as the semantics defined in this document are not changed.

8.4. Client-Based Pseudonyms

Previous sections have discussed requesting pseudonyms based on registered identities. In some cases a requestor desires a pseudonym to be issued using *ad hoc* data that is specifies as an extension to the RST request. As with all WS-Trust parameters, the IP/STS is NOT REQUIRED to honor the parameter, but if it does, it SHOULD echo the parameter in the RSTR.

A requestor MAY specify the `<fed:ClientPseudonym>` element to indicate pseudonym information it would like used in the issued token. The STS MUST accept all of the information or none of it. That is, it MUST NOT use some pseudonym information but not other pseudonym information.

The syntax of the `<fed:ClientPseudonym>` element is as follows:

```
<fed:ClientPseudonym ...>
  <fed:PPID ...>xs:string</fed:PPID> ?
  <fed:DisplayName ...>xs:string</fed:DisplayName> ?
  <fed:Email ...>xs:string</fed:EMail> ?
  ...
</fed:ClientPseudonym>
```

/fed:ClientPseudonym

This indicates a request to use specific identity information in resulting security tokens.

/fed:ClientPseudonym/fed:PPID

If the resulting security token contains any form of private personal identifier, this string value is to be used as the basis. The issuer MAY use this value as the input (a seed) to a custom function and the result used in the issued token.

/fed:ClientPseudonym/fed:PPID/@{any}

This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:ClientPseudonym/fed:DisplayName

If the resulting security token contains any form of display or subject name, this string value is to be used.

/fed:ClientPseudonym/fed:DisplayName/@{any}

This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:ClientPseudonym/fed:EMail

If the resulting security token contains any form electronic mail address, this string value is to be used.

/fed:ClientPseudonym/fed:Email/@{any}

This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:ClientPseudonym/{any}
This extensibility point allows other pseudonym information to be specified. If the STS does not understand any element it MUST either ignore the entire `<fed:ClientPseudonym>` or Fault.

/fed:ClientPseudonym/@{any}
This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

8.5. Indicating Freshness Requirements

There are times when a token requestor desires to limit the age of the credentials used to authenticate. The parameter MAY be specified in a RST to indicate the desired upper bound on credential age. As well this parameter is used to indicate if the requestor is willing to allow issuance based on cached credentials.

The syntax of the `<fed:Freshness>` element is as follow:

```
<fed:Freshness AllowCache="xs:boolean" ...>
  xs:unsignedInt
</fed:Freshness>
```

/fed:Freshness
This indicates a desire to limit the age of authentication credentials. This required unsigned integer value indicates the upper bound on credential age specified in minutes only. A value of zero (0) indicates that the STS is to immediately verify identity if possible or use the minimum age credentials possible if immediate (e.g. interactive) verification is not possible. If the `AllowCache` attribute is specified, then the cached credentials SHOULD meet the freshness time window.

/fed:Freshness/@{any}
This extensibility mechanism allows additional attributes to be specified so long as they do not alter the semantics defined in this specification.

/fed:Freshness/@AllowCache
This OPTIONAL Boolean qualifier indicates if cached credentials are allowed. The default value is *true* indicating that cached information MAY be used. If *false* the STS SHOULD NOT use cached credentials in processing the request.

If the credentials provided are valid but do not meet the freshness requirements, then the `fed:NeedFresherCredentials fault` MUST be returned informing the requestor that they need to obtain fresher credentials in order to process their request.

9. Authorization

An authorization service is a specific instance of a security token service (STS). To ensure consistent processing and interoperability, this specification defines a common model for authorization services, a set of extensions enabling rich authorization, and a common profile of [WS-Trust] to facilitate interoperability with authorization services.

This section describes a model and two extensions specific to rich authorization. The first allows additional context information to be provided in authorization requests. The second allows services to indicate that additional claims are required to successfully process specific requests.

9.1. Authorization Model

An authorization service is an STS that operates in a decision brokering process. That is, it receives a request (either directly or on behalf of another party) for a token (or set of tokens) to access another service. Such a service MAY be separate from the target service or it MAY be co-located. The authorization service determines if the requested party can access the indicated service and, if it can, issues a token (or set of tokens) with the allowed rights at the specified service. These two aspects are distinct and could be performed by different collaborating services.

In order to make the authorization decision, the authorization service must ensure that the requestor has presented and proven the claims required to access the target service (or resource) indicated in the request (e.g. in the `<wsp:AppliesTo>` parameter). Logically, the authorization service constructs a table of name/value pairs representing the claims required by the target service. The logical *requirement table* is constructed from the following sources and may be supplemented by additional service resources:

- The address of the EPR for the target service
- The reference properties from the EPR of the target service
- Parameters of the RST
- External access control policies

Similarly, the claim table is a logical table representing the claims and information available for the requestor that the authorization service uses as the basis for its decisions. This logical table is constructed from the following sources:

- Proven claims that are bound to the RST request (both primary and supporting)
- Supplemental authorization context information provided in the request
- External authorization policies

9.2 Indicating Authorization Context

In the [WS-Trust] protocol, the requestor of a token conveys the desired properties of the required token (such as the token type, key type, claims needed, etc.) in the token request represented by the RST element. Each such property is represented by a child element of the RST, and is typically specified by the Relying Party that will consume the issued token in its security policy assertion as defined by [WS-SecurityPolicy]. The token properties specified in a token request (RST) generally translate into some aspect of the content of the token that is issued by a STS. However, in many scenarios, there is a need to be able to convey additional contextual data in the token request that influences the processing and token issuance behavior at the STS. The supplied data may or may not directly translate into some aspect of the actual token content.

To enable this a new element, `<auth:AdditionalContext>`, is defined to provide additional context information. This MAY be specified in RST requests and MAY be included in RSTR responses.

The syntax is as follows:

```
<wst:RequestSecurityToken>
  ...
  <auth:AdditionalContext>
    <auth:ContextItem Name="xs:anyURI" Scope="xs:anyURI" ? ...>
```

```
      (<auth:Value>xs:string</auth:Value> |
       xs:any ) ?
    </auth:ContextItem> *
    ...
  </auth:AdditionalContext>
  ...
</wst:RequestSecurityToken>
```

The following describes the above syntax:

…/auth:AdditionalContext

This optional element provides additional context for the authorization decision (which determines token issuance).

…/auth:AdditionalContext/ContextItem

This element is provides additional authorization context as simple name/value pairs. Note that this is the only `fed:AdditionalContext` element defined in this specification.

…/auth:AdditionalContext/ContextItem/@Name

This required URI attribute specifies the kind of the context item being provided.

There are no pre-defined context names.

…/auth:AdditionalContext/ContextItem/@Scope

This OPTIONAL URI attribute specifies the scope of the context item. That is, the subject of the context item. If this is not specified, then the scope is undefined.

The following scopes a pre-defined but others MAY be added:

URI	**Description**
`http://schemas.xmlsoap.org/ws/2006/12/authorization/ctx/requestor`	The context item applies to the requestor of the token (or the OnBehalfOf)
`http://schemas.xmlsoap.org/ws/2006/12/authorization/ctx/target`	The context item applies to the intended target (AppliesTo) of the token

URI	Description
http://schemas.xmlsoap.org/ws/2006/12/authorization/ctx/action	The context item applies to the intended action at the intended target (AppliesTo) of the token

…/auth:AdditionalContext/ContextItem/Value

This OPTIONAL string element specifies the simple string value of the context item.

…/auth:AdditionalContext/ContextItem/{any}

This OPTIONAL element allows a custom context value to be associated with the context item. This CANNOT be specified along with the Value element (there can only be a single value).

…/auth:AdditionalContext/ContextItem/@{any}

This extensibility point allows additional attributes to be specified so long as they do not violate any semantics defined in this document.

…/auth:AdditionalContext/@{any}

This extensibility point allows additional attributes to be specified so long as they do not violate any semantics defined in this document.

…/auth:AdditionalContext/{any}

This element has an open content model allowing different types of context to be specified. That is, custom elements can be defined and included so long as all involved parties understand the elements.

An example of an RST token request where this element is used to specify additional context data is given below. Note that this example specifies claims using a custom dialect.

```
<wst:RequestSecurityToken>
  <wst:TokenType>
      urn:oasis:names:tc:SAML:1.0:assertion
  </wst:TokenType>
  <wst:RequestType>
    http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
  </wst:RequestType>
  <wst:Claims Dialect="...">
    ...
  </wst:Claims>
  ...
```

```
  <auth:AdditionalContext>
    <auth:ContextItem Name="urn:...:PurchaseAmount">
      <auth:Value>125.00</auth:Value>
    </auth:ContextItem>
    <auth:ContextItem Name="urn:...:MerchantId">
      <auth:Value>FABRIKAM 92305645883256</auth:Value>
    </auth:ContextItem>
  </auth:AdditionalContext>
</wst:RequestSecurityToken>
```

9.3 Common Claim Dialect

There are different claim representations that are used across different Web Service implementations making it difficult to express claims in a general way. To facilitate interoperability, this section describes a simple dialect for expressing basic claims in a format-neutral way. This new dialect is represented by the following URI:

```
http://schemas.xmlsoap.org/ws/2006/12/authorization/authclaims
```

This dialect is used within the `<wst:Claims>` element when making token requests (and MAY be included in RST responses). The syntax is as follows:

```
<wst:RequestSecurityToken>
  ...
  <wst:Claims
Dialect="http://schemas.xmlsoap.org/ws/2006/12/authorization/authclaims">
    <auth:ClaimType Uri="xs:anyURI" Optional="xs:boolean">
      (<auth:Value>...</auth:Value> |
       xs:any) ?
    </auth:ClaimType> *
  </wst:Claims>
  ...
</wst:RequestSecurityToken>
```

The following describes the above syntax:

…/auth:ClaimType

This element represents a specific claim.

…/auth:ClaimType/@Uri

This required URI attribute specifies the kind of the claim being indicated. The following claim type is pre-defined, but other types MAY be defined:

URI	Description
`http://schemas.xmlsoap.org/ws/2006/12/authorization/claims/action`	The `wsa:Action` specified in a request

…/auth:ClaimType/@Optional

This optional boolean attribute specifies the claim is optional (true) or required (false).

…/auth:ClaimType/auth:Value

This OPTIONAL element allows a specific string value to be specified for the claim.

…/auth:ClaimType/@{any}

This extensibility point allows additional attributes to be specified so long as they do not violate any semantics defined in this document.

…/auth:ClaimType/{any}

This extensibility point allows additional values types to be specified for the claim.

9.4. Authorization Requirements

Authorization requestors and issuing services (providers) compliant with this specification MUST conform to the rules described in this section when issuing RST requests and returning RSTR responses.

R001 – The authorization service MUST accept an `<wsp:AppliesTo>` target in the RST

R002 – The authorization service MUST specify an `<wsp:AppliesTo>` target in the RSTR if one is specified in the RST

R003 – The authorization service SHOULD encode the `<wsp:AppliesTo>` target in issued tokens if the token format supports it

R004 – The `<wsp:AppliesTo>` target for issued token MAY be for a broader scope than the scope specified in the RST but MUST NOT be narrower (as specified in WS-Trust)

R005 – The authorization service MUST accept reference properties in the `<wsp:AppliesTo>` target

R006 – The authorization service MUST accept the `<auth:AdditionalContext>` parameter

R007 – The authorization service MUST accept the claim dialect defined in this specification

R008 – The authorization service MAY ignore elements in the *auth*`:AdditionalContext` parameter if it doesn't recognize or understand them

10. Indicating Specific Policy/Metadata

When a requestor communicates with a recipient service there may be additional security requirements, beyond those in the general security policy or other metadata, that are required based on the specifics of the request. For example, if a request contains a "gold customer" custom message header to indicate customer classification (and routing), then proof that the requestor is a gold member may be required when the request is actually authorized. There may also be contextual requirements which are hard to express in a general policy. For example, if a requestor wants to submit a purchase, it may be required to present a token from a trusted source attesting that the requestor has the requisite funds.

To address this scenario a mechanism is introduced whereby the recipient service can indicate to the requestor that additional security semantics apply to the request. The requestor MAY reconstruct the message in accordance with the new requirements if it can do so. In some cases the requestor may need to obtain additional tokens from an authorization or identity service and then reconstruct and resubmit the message.

The mechanism for dynamically indicating that a specific policy or metadata applies to a specific request is to issue a specialized SOAP Fault. This fault indicates to the requestor that additional security metadata is required. The new metadata, in its complete form (not a delta) is specified in the fault message using the WS-MetadataExchange format.

The fault is the `fed:SpecificMetadata` and is specified as the fault code. The `<S:Detail>` of this fault contains a `mex:Metadata` element containing sections with the effective metadata for the endpoint processing this specific request.

The following example illustrates a fault with embedded policy:

```
<S:Envelope xmlns:S="..." xmlns:auth="..." xmlns:wst="..." xmlns:fed="..."
      xmlns:sp="..." xmlns:wsp="..." xmlns:mex="...">
  <S:Body>
    <S:Fault>
      <S:Code>
        <S:Value>fed:SpecificMetadata</S:Value>
      </S:Code>
      <S:Reason>
        <S:Text>Additional credentials required in order to
                perform operation. Please resubmit request with
                appropriate credentials.
        </S:Text>
      </S:Reason>
      <S:Detail>
       <mex:Metadata>
        <mex:MetadataSection
                 Dialect='http://schemas.xmlsoap.org/ws/2004/09/policy'>
         <wsp:Policy>
          ...
           <sp:EndorsingSupportingTokens>
            <sp:IssuedToken>
              <sp:Issuer>...</sp:Issuer>
              <sp:RequestSecurityTokenTemplate>
                <wst:Claims>
                  ...
                </wst:Claims>
```

```
                <auth:AdditionalContext>
                 ...
                </auth:AdditionalContext>
               ...
              </sp:RequestSecurityTokenTemplate>
            </sp:IssuedToken>
          </sp:EndorsingSupportingTokens>
        </wsp:Policy>
       </mex:MetadataSection>
      </mex:Metadata>
     </S:Detail>
   </S:Fault>
  </S:Body>
</S:Envelope>
```

11. Authentication Types

The [WS-Trust] specification defines the `wst:AuthenticationType` parameter to indicate a desired type of authentication (or to return the type of authentication verified). However, no pre-defined values are specified. While any URI can be used, to facilitate federations the following optional types are defined but are NOT REQUIRED to be used:

URI	***Description***
`http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/unknown`	Unknown level of authentication
`http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/default`	Default sign-in mechanisms
`http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/Ssl`	Sign-in using SSL
`http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/SslAndKey`	Sign-in using SSL and a security key
`http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/SslAndStrongPasssword`	Sign-in using SSL and a “strong” password
`http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/SslAndStrongPasswordWithExpiration`	Sign-in using SSLand a “strong” password with expiration
`http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/smartcard`	Sign-in using Smart Card

12. Privacy

When a requestor contacts an authority to obtain a security token or to obtain authorization for an action it is often the case that information subject to personal or organizational privacy requirements may be presented in order to authorize the request. In such cases the authority may require the requestor to indicate the restrictions it expects on the use and distribution of sensitive information contained in tokens it obtains. In this document, this is referred to as a "disclosure constraint". It should be noted that disclosure constraints may apply if the requestor is requesting tokens for itself or if the requestor is acting on behalf of another party.

This specification describes how requestors can communicate their disclosure constraints to security token services using the [WS-Trust] protocol. It additionally facilitates the requestor's compliance with such constraints by allowing it to request elevated data protection for some or all of the response and issued tokens. The disclosure constraint and protection elevation request are communicated using existing WS-Trust mechanisms as well as extensions defined in this specification.

The WS-Trust specification describes how to request tokens as well as parameters to the token request (RST) for indicating how to encrypt proof information as well as algorithms to be used. The following sub-sections define extension parameters that MAY be specified in RST requests (and echoed in RSTR responses) to indicate additional privacy options which complement the existing WS-Trust parameters.

12.1 Confidential Tokens

The information contained within an issued token may be confidential or sensitive. Consequently, the requestor may wish to have this information protected (confidential) so that only the intended recipient of the resulting token (or tokens) can access the information.

The [WS-Trust] specification describes how to indicate a key to use if any data in the token is to be encrypted, but doesn't specify any mandates around when or what data is to be protected. This parameter indicates a protection requirement from the requestor (the STS MAY choose to protect data even if the requestor doesn't mandate it).

Any protected (encrypted) information is secured using the token specified in the `<wst:Encryption>` parameter or using a token the recipient knows to be correct for the request.

The following parameters MAY be specified in an RST request (and echoed in an RSTR response) to indicate that potentially sensitive information in the token be protected:

```
<wst:RequestSecurityToken>
  ...
  <priv:ProtectData ...>
    <wst:Claims ...>...</wst:Claims> ?
    ...
  </priv:ProtectData>
  ...
</wst:RequestSecurityToken>
```

The following describes the above syntax:

…/priv:ProtectData

This optional parameter indicates that sensitive information in any resultant tokens MUST be protected (encrypted). If specific claims are identified they MUST be protected. The issuer may have an out-of-band agreement with the requestor as to what must be protected. If not, and if specific claims are not identified, the issuer MUST protect all claims. The issuer MAY choose to protect more than just the requested claims.

…/priv:ProtectData/@{any}

This extensibility point allows additional attributes to be specified so long as they do not violate any semantics defined in this document.

…/priv:ProtectData/wst:Claims

This allows the requestor to indicate specific claims which, at a minimum, MUST be protected. This re-uses the claim specification mechanism from [WS-Trust]. Claims specified in this set MUST be protected. There is no requirement that all claims specified are in the issued token. That is, claims identified but not issued MAY be ignored by the STS.

…/priv:ProtectData/{any}

This extensibility point allows additional content to be specified so long as it does not violate any semantics defined in this document.

12.2 Parameter Confirmation

The RST request MAY contain a number of parameters indicating a requestor's disclosure constraints and data protection preferences. The STS is not required to honor these preferences and may, or may not, include selected parameters in any RSTR response.

For privacy reasons a requestor may wish to (a) know if privacy preferences (or any RST parameter) were accepted or not, (b) what default parameter values were used, (c) require that privacy preferences (or any RST parameter) be honored, and (d) know what the STS is reporting in a token if it is protected and unreadable by the requestor.

The following parameters MAY be specified in a RST request (and echoed in an RSTR response) to indicate to support these requirements:

```
<wst:RequestSecurityToken>
 ...
 <priv:EnumerateParameters ...>
   <xs:list itemType='xs:QName' />
 </priv:EnumerateParameters>
 <priv:FaultOnUnacceptedRstParameters ...>
   ...
</priv:FaultOnUnacceptedRstParameters>
 <priv:EnumerateAllClaims ...>
 ...
 <priv:EnumerateAllClaims ...>
 ...
</wst:RequestSecurityToken>
```

The following describes the above syntax:

…/priv:EnumerateParameters

A RST request MAY include parameters but the STS is not required to honor them. As such there is no way for the requestor to know what values where used by the STS. This optional parameter provides a way to request the STS to return the values it used for parameters (or Fault if it refuses) – either taken from the RST or defaulted using internal policy or settings. The contents of this parameter indicate a list of QNames that represents RST parameters which MUST be included in the RSTR. That is, each QName listed MUST be present in the RSTR returned by the STS indicating the value the STS used for the parameter.

…/priv:EnumerateParameters/@{any}

This extensibility point allows additional attributes to be specified so long as they do not violate any semantics defined in this document.

…/priv:FaultOnUnacceptedRstParameters

This optional parameter indicates that if any parameters specified in the RST are not accepted by the STS, then the STS MUST Fault the request (see the Error Code section for the applicable Fault code). This means that any unknown parameter causes the request to fail. Note that this includes extension parameters to the RST.

…/priv:FaultOnUnacceptedRstParameters/@{any}

This extensibility point allows additional attributes to be specified so long as they do not violate any semantics defined in this document.

…/priv:FaultOnUnacceptedRstParameters/{any}

This extensibility point allows additional content to be specified so long as it does not violate any semantics defined in this document.

…/priv:EnumerateAllClaims

This optional parameter indicates that all claims issued in resulting tokens MUST be identified in the RSTR so that the requestor can inspect them. The claims are returned in a `<wst:Claims>` element in the RSTR.

…/priv:EnumerateAllClaims/@{any}

This extensibility point allows additional attributes to be specified so long as they do not violate any semantics defined in this document.

…/priv:EnumerateAllClaims/{any}

This extensibility point allows additional content to be specified so long as it does not violate any semantics defined in this document.

12.3 Privacy Statements

Some services offer privacy statements. This specification defines a mechanism by which privacy statements, in any form of representation, can be obtained using the mechanisms defined in [WS-Transfer/WS-ResourceTransfer].

The following URI is defined which can be used as a metadata section dialect in [WS-Transfer/WS-ResourceTransfer]:

```
http://schemas.xmlsoap.org/ws/2006/12/privacy/privacypolicy
```

As well, the following element can be used to indicate the EPR to which a [WS-Transfer/WS-ResourceTransfer] GET message can be sent to obtain the privacy policy:

```
<priv:PrivacyPolicyEndpoint SupportsMex="xs:boolean" ?>
  ...endpoint reference value...
</priv:PrivacyPolicyEndpoint
```

This element is an endpoint-reference as described in [WS-Addressing]. A [WS-Transfer/WS-ResourceTransfer] GET message can be sent to it to obtain the previously defined privacy policy dialect. If the `SupportsMex` attribute is true (the default is false), then a [WS-MetadataExchange] request can be directed at the endpoint.

Note that no specific privacy policy form is mandated so requestors must inspect the contents of the returned privacy policy (or policies) to determine if they can process it (them). The privacy policy could be a complete privacy policy document, a privacy policy document that references other privacy policies, or even a compact form of a privacy policy. The form of these documents is outside the scope of this document.

Alternatively, HTTP GET targets can be specified by including a URL with the following federated metadata statement:

```
<priv:PrivacyNoticeAt ...> location URL </priv:PrivacyNoticeAt>
```

13. Web (Passive) Requestors

This specification defines a model and set of messages for brokering trust and federation of identity and authentication information across different trust realms and protocols. This section describes how this Federations model is applied to Web requestors such as Web browsers that cannot directly make Web Service requests.

13.1. Approach

The federation model previously described builds on the foundation established by [WS-Security] and [WS-Trust]. Typical Web client requestors cannot perform the message security and token request operations defined in these specifications. Consequently, this section describes the mechanisms for requesting, exchanging, and issuing security tokens within the context of a Web requestor.

Web requestors use different but philosophically compatible message exchanges. For example, the resource might act as its own Security Token Service (STS) and not use a separate service (or even URI) thereby eliminating some steps. It is expected that subsequent profiles can be defined to extend the Web mechanisms to include additional exchange patterns.

13.1.1. Sign-On

The primary issue for *Web browsers* is that there is no easy way to directly issue SOAP requests. Consequently, the processing must be performed within the confines of the base HTTP 1.1 functionality (GET, POST, redirects, and cookies) and conform as closely as possible to the WS-Trust protocols for token acquisition.

At a high-level, requestors are associated with an Identity Provider (IP) or Security Token Service (STS) where they authenticate themselves. At the time/point of initial authentication an artifact/cookie MAY be created for the requestor at their Identity Provider so that every request for a resource doesn't require requestor intervention. At other times, authentication at each request is the desired behavior.

In the Web approach, there is a common pattern used when communicating with an IP/STS. In the first step, the requestor accesses the resource; the requestor is then redirected to an IP/STS if no token or cookie is supplied on the request. The requestor may be redirected to a local IP/STS operated by the resource provider. If it has not cached data indicating that the requestor has already been authenticated, a second redirection to the requestor's IP/STS will be performed. This redirection process may require prompting the user to

determine the requestor's home realm. The IP/STS in the requestor's home realm generates a security token for use by the federated party. This token may be consumed directly by the resource, or be exchanged at the resource's IP/STS for a token consumable by the resource. In some cases the requestor's IP/STS has the requisite information cached to be able to issue a token, in other cases it must prompt the user. Note that the resource's IP/STS can be omitted if the resource is willing to consume the requestor's token directly.

The figure below illustrates an example flow where there is no resource IP/STS. As depicted, all communication occurs with the standard HTTP GET and POST methods, using redirects (steps 2→3 and 5→6) to automate the communication. Note that when returning non-URL content a POST is required (e.g. in step 6) if a result reference is not used. In step 2 the resource may act as its own IP/STS so communication with an additional service isn't required. Note that step 3 depicts the resource redirecting directly to the requestor's IP/STS. As previously discussed, this could redirect to an IP/STS for the resource (or any number of chained IP/STS services). It might also redirect to a home realm discovery service.

It should be noted that in step 4, the authentication protocol employed MAY be implementation-dependent.

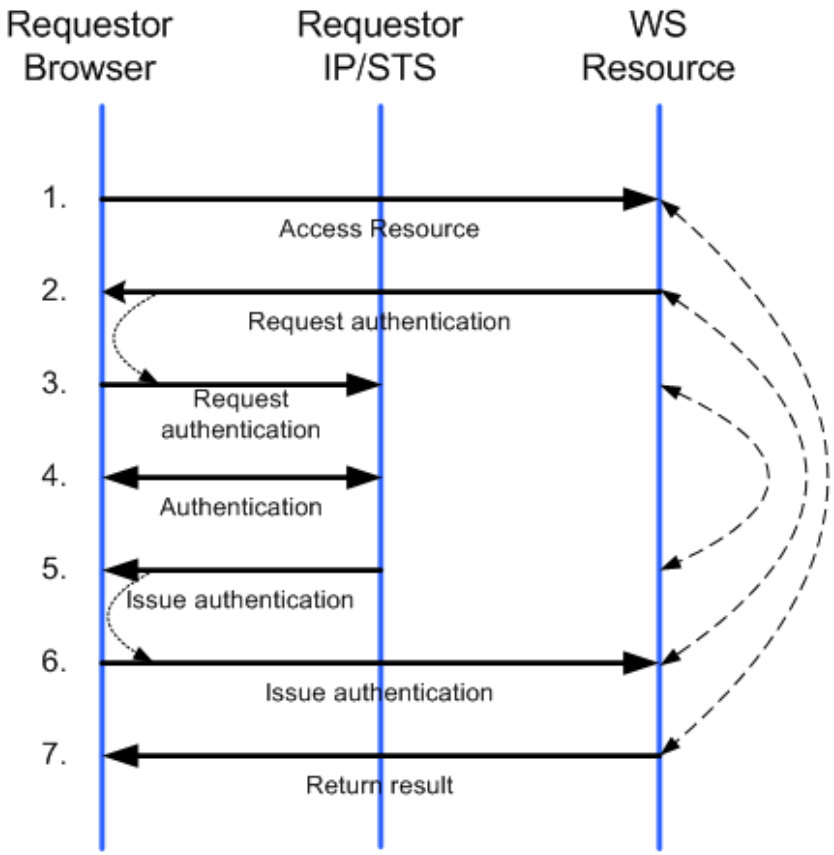


Figure 25: Sample Browser Sign-On

13.1.2. Sign-Out

For Web browsers, sign-out can be initiated by selecting the sign-out URL at a resource. In doing so, the browser will ultimately be redirected to the requestor's IP/STS indicating sign-out. Note that the browser MAY be first redirected to the resource's IP/STS and then to the

requestor's IP/STS. Note that if multiple IP/STS services are used, and unaware of each other, multiple sign-outs may be required.

The requestor's IP/STS SHOULD keep track of the realms to which it has issued tokens where cleanup may be required – specifically the IP/STS for the realms (or resources if different). When the sign-out is received at the requestor's IP/STS, it SHOULD initiate clean-up (e.g. issuing HTTP GET requests against the tracked realms indicating a sign-out cleanup is in effect or it can use the sign-out mechanism previously discussed). The exact mechanism by which this occurs is up to the IP/STS and is policy-driven. The only requirement is that a sign-out cleanup be performed at the IP/STS so that subsequent requests to the IP/STS don't use cached data.

As described in section 4.2, there are two possible flows for these messages. They could be effectively chained through all the STSs involved in the session by successively redirecting the browser between each resource IP/STS and the requestor's IP/STS. Or the requestor's IP/STS can send sign-out messages to all the other STSs in parallel. The chained (sequential) approach has been found to be fragile in practice. If a resource IP/STS fails to redirect the user after cleaning up local state, or the network partitions, the sign-out notification will not reach all the resource IP/STSs involved. For this reason, compliant implementations SHOULD employ the parallel approach.

When a sign-out clean-up GET is received at a realm, the realm SHOULD clean-up any cached information and delete any associated artifacts/cookies. If requested, on completion the requestor is redirected back to requestor's IP/STS.

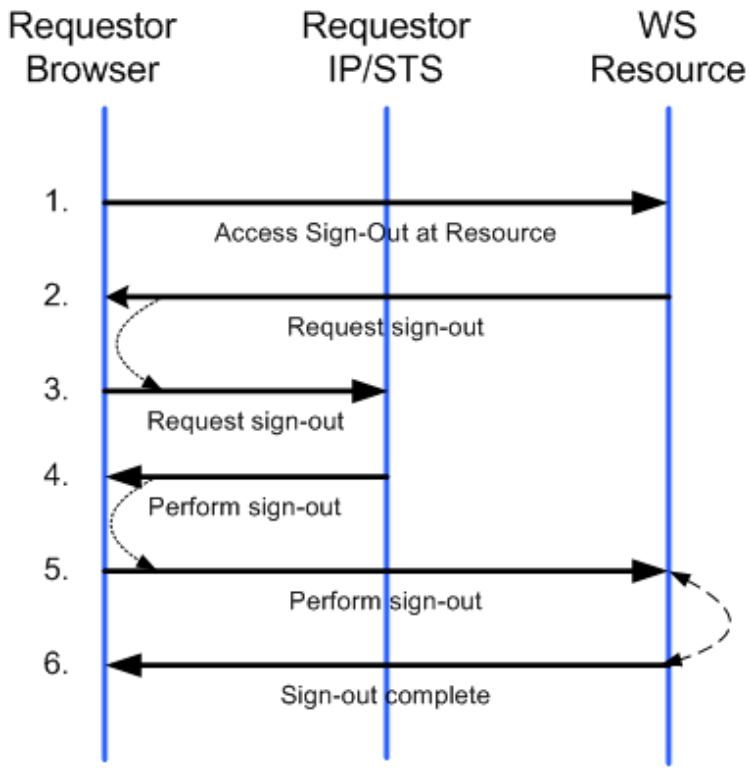


Figure 26: Sample Browser Sign-Out

The figure above illustrates this process where a resource-specific IP/STS doesn't exist. The mechanism illustrated use redirection in steps 2 and 4 (optional) and the general *correlation*

of messages to chain the sign-out. As previously noted there could be a resource-specific IP/STS which handles local chaining or notification.

It should be noted that as a result of the single sign-out request (steps 5 and 6), an IP/STS MAY send sign-out messages as described in this specification.

13.1.3. Attributes

At a high-level, attribute processing uses the same mechanisms defined for security token service requests and responses. That is, redirection is used to issue requests to attribute services and subsequent redirection returns the results of the attribute operations. All communication occurs with the standard HTTP 1.1 GET and POST methods using redirects to automate the communication as shown in the example below.

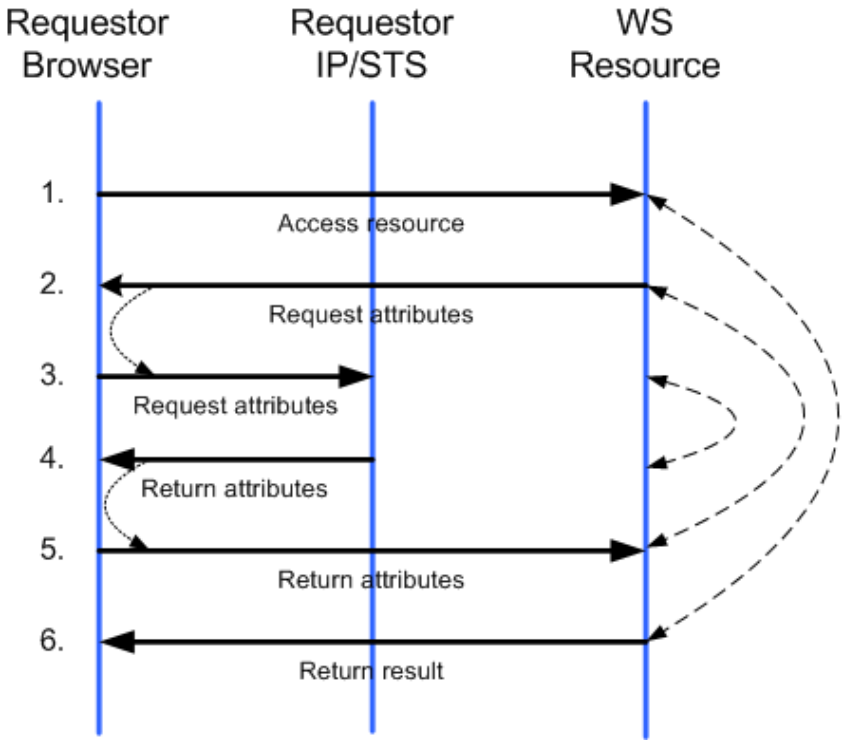


Figure 27: Sample Browser Attribute Access

The figure above illustrates this process including calling out the redirection in steps 2 and 4 and the general *correlation* of messages for an attribute scenario where there is no resource-specific IP/STS.

As well, it should be noted that as a result of step 3 the IP/STS MAY prompt the user for approval before proceeding to step 4.

13.1.4. Pseudonyms

At a high-level, pseudonym processing uses the same mechanisms defined for attribute and security token service requests. That is, redirection is used to issue requests to pseudonym services and subsequent redirection returns the results of the pseudonym operations. All communication occurs with the standard HTTP GET and POST methods using redirects to automate the communication as in the example below.

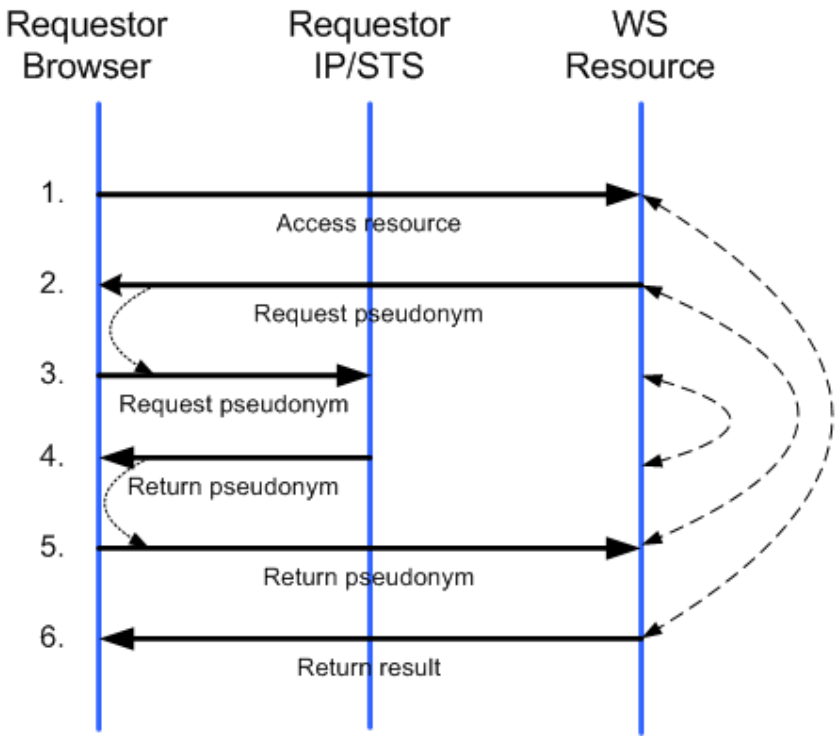


Figure 28: Sample Browser Pseudonym Access

The figure above illustrates this process including calling out the redirection in steps 2 and 4 and the general *correlation* of messages for an attribute scenario where there is no resource-specific IP/STS.

13.1.5. Artifacts/Cookies

In order to prevent requestor interaction on every request for security token, artifacts/cookies can be used by SSO implementations as they are used today to cache state and/or authentication information or issued tokens. However implementations MAY omit this caching if the desired behavior is to authenticate on every request. As noted in the Security Consideration section later in this document, there are security issues when using cookies.

There are no restrictions placed on artifacts/cookie formats – they are up to each service to determine. However, it is RECOMMENDED artifacts/cookies be encrypted or computationally hard to compromise.

13.1.6. Bearer Tokens and Token References

In cases where bearer tokens or references to tokens are passed it is strongly RECOMMENDED that the messages use transport security in order to prevent attack.

13.1.7. Freshness

In cases where a resource requires specific authentication freshness, they can specify requirements in their IP/STS requests, as described in the following section (see 13.2.2).

13.2. HTTP Protocol Syntax

This section describes the syntax of the protocols used by Web requestors. This protocol typically uses the redirection facilities of HTTP 1.1. This happens using a standard HTTP 302 error code for redirects (as illustrated below) and HTTP POST to push the forms:

```
HTTP/1.1 302 Found
Location: url?parameters
```

The exact parameters and form fields are described in detail in the sub-sections that follow the detailed example.

In the descriptions below, some mechanisms are optional meaning they MAY be supported. Within a mechanism, certain parameters MUST be specified while others, noted using square brackets, are optional and MAY or MAY NOT be present.

13.2.1. Parameters

All HTTP 1.1 methods (both GET and POST) used in the redirection protocol allow query string parameters as illustrated below:

```
GET url?parameters
POST url?parameters
```

The GET and POST requests have required parameters and may have optional parameters depending on the operation being performed. For GET requests, these parameters are specified in the query string; for POST requests, these parameters are specified in the POST body (using the standard encoding rules for POST). The query string parameters of a POST request SHOULD be for extensibility only, and MAY be ignored by an implementation that is otherwise compliant with this specification.

The following describes the parameters used for messages in this profile:

```
wa=string
[wreply=URL]
[wres=URL]
[wctx=string]
[wp=URI]
[wct=timestring]
[wfed=string]
```

wa
: This required parameter specifies the action to be performed. By including the action, URIs can be overloaded to perform multiple functions. For sign-in, this string MUST be "wsignin1.0". Note that this serves roughly the same purpose as the WS-Addressing Action header for the WS-Trust SOAP RST messages.

wreply
: This optional parameter is the URL to which responses are directed. Note that this serves roughly the same purpose as the WS-Addressing `<wsa:ReplyTo>` header for the WS-Trust SOAP RST messages.

wres
: This optional parameter is the URL for the resource accessed. This is a legacy parameter which isn't typically used. The *wtrealm* parameter is typically used instead.

wctx

This optional parameter is an opaque context value that MUST be returned with the issued token if it is passed in the request. Note that this serves roughly the same purpose as the WS-Trust SOAP RST @Context attribute. In order not to exceed URI length limitations, the value of this parameter should be as small as possible.

wp

This optional parameter is the URL for the policy which can be obtained using an HTTP GET and identifies the policy to be used related to the action specified in "wa", but MAY have a broader scope than just the "wa". Refer to WS-Policy and WS-Trust for details on policy and trust. This attribute is only used to reference policy documents. Note that this serves roughly the same purpose as the Policy element in the WS-Trust SOAP RST messages.

wct

This optional parameter indicates the current time at the sender for ensuring freshness. This parameter is the string encoding of time using the XML Schema datetime time using UTC notation. Note that this serves roughly the same purpose as the WS-Security Timestamp elements in the Security headers of the SOAP RST messages.

wfed

This optional parameter indicates the federation context in which the request is made. This is equivalent to the `FederationId` parameter in the RST message.

Note that any values specified in parameters are subject to encoding as specified in the HTTP 1.1 specification.

When an HTTP POST is used, any of the query strings can be specified in the form contents using the same name. Note that in this profile form values take precedence over URL parameters.

Parameterization is extensible so that cooperating parties can exchange additional information in parameters based on agreements or policy.

13.2.2. Requesting Security Tokens

The HTTP requests to an Identity Provider or security token service use a common syntax based on HTTP forms. Requests typically arrive using the HTTP GET method as illustrated below but MAY be issued using a POST method:

```
GET resourceSTS?parameters HTTP/1.1

POST resourceSTS?parameters HTTP/1.1
```

The parameters described in the previous section (wa, wreply, wres, wctx, wp, wct) apply to the token request. The additional parameters described below also apply. Note that any values specified in forms are subject to encoding as described in the HTTP 1.1 specification.

The following describes the additional parameters used for a token request:

```
wtrealm=string

[wfresh=freshness]

[wauth=uri]

[wreq=xml]
```

wtrealm

This required parameter is the URI of the requesting realm. The wtrealm SHOULD be the security realm of the resource in which nobody (except the resource or authorized

delegates) can control URLs. Note that this serves roughly the same purpose as the AppliesTo element in the WS-Trust SOAP RST messages.

wfresh
: This optional parameter indicates the freshness requirements. If specified, this indicates the desired maximum age of authentication specified in minutes. An IP/STS SHOULD not issue a token with a longer lifetime. If specified as "0" it indicates a request for the IP/STS to re-prompt the user for authentication before issuing the token. Note that this serves roughly the same purpose as the Freshness element in the WS-Trust SOAP RST messages.

wauth
: This optional parameter indicates the required authentication level. Note that this parameter uses the same URIs and is equivalent to the `wst:AuthenticationType` element in the WS-Trust SOAP RST messages.

wreq
: This optional parameter specifies a token request using either a `<wst:RequestSecurityToken>` element or a full request message as described in WS-Trust. If this parameter is not specified, it is assumed that the responding service *knows* the correct type of token to return. Note that this can contain the same RST payload as used in WS-Trust RST messages.

To complete the protocol for requesting a token, it is necessary to redirect the Web requestor from the resource, or its local IP/STS, to the requestor's IP/STS. Determining the location of this IP/STS is frequently referred to as Home Realm Discovery; that is, determining the realm which manages the requestor's identity and thus where its IP/STS is located. This frequently involves interaction with the user (see section 13.5 for additional discussion). There are situations – particularly when users only access resources via portals and never directly via bookmarked URLs – when it can be advantageous to include the requestor's home realm in the request to avoid the requirement for human interaction. The following parameter MAY be specified for this purpose.

```
[whr=string]
```

whr
: This optional parameter indicates the account partner realm of the client. This parameter is used to indicate the IP/STS address for the requestor. This may be specified directly as a URL or indirectly as an identifier (e.g. urn: or uuid:). In the case of an identifier the recipient is expected to know how to translate this (or get it translated) to a URL. When the *whr* parameter is used, the resource, or its local IP/STS, typically removes the parameter and writes a cookie to the client browser to remember this setting for future requests. Then, the request proceeds in the same way as if it had not been provided. Note that this serves roughly the same purpose as federation metadata for discovering IP/STS locations previously discussed.

In the event that the XML request cannot be passed in the form (due to size or other considerations), the following parameter MAY be specified and the form made available by reference:

```
wreqptr=url
```

wreqptr
: This optional parameter specifies a URL for where to find the request (`wreq` parameter). Note that this does not have a WS-Trust parallel.

When using wreqptr it is strongly RECOMMENDED that the provider of the wreqptr data authenticate the data to the consumer (relying party) in some way and that the provider authenticate consumers requesting the wreqptr data. If the wreqptr data is sensitive the provider SHOULD consider ensuring confidentiality of the data transfer.

The RST is logically constructed to process the request. If one is specified either directly or indirectly via wreqptr it is the authoritative source or parameter information. That is, parameters outside of the RST (e.g. wreq, wfresh, wtrealm, …) are used to construct an RST if the RST is not present or if the corresponding RST values are not present.

13.2.3. Returning Security Tokens

Security tokens are returned by passing an HTTP form. To return the tokens, this profile embeds a `<wst:RequestSecurityTokenResponse>` element as specified in [WS-Trust].

```
POST resourceURI?parameters HTTP/1.1

GET resourceURI?parameters HTTP/1.1
```

In many cases the IP/STS to whom the request is being made, will prompt the requestor for information or for confirmation of the receipt of the token. As a result, the IP/STS can return an HTTP form to the requestor who then submits the form using an HTTP POST method. This allows the IP/STS to return security token request responses in the body rather than embedded in the limited URL query string. However, in some circumstances interaction with the requestor may not be required (e.g. cached information). In these circumstances the IP/STS have several options:

1. Use a form anyway to confirm the action
2. Return a form with script to automate and instructions for the requestor in the event that scripting has been disabled
3. Use HTTP GET and return a pointer to the token request response (unless it is small enough to fit inside the query string)

This specification RECOMMENDS using the POST method as the GET method requires additional state to be maintained and complicates the cleanup process whereas the POST method carries the state inside the method.

Note that when using the POST method, any values specified in parameters are subject to encoding as described in the HTTP 1.1 specification. The standard parameters apply to returning tokens as do the following additional form parameters:

```
wresult=xml

[wctx=string]
```

wresult
: This required parameter specifies the result of the token issuance. This can take the form of the `<wst:RequestSecurityTokenResponse>` element or `<wst:RequestSecurityTokenResponseCollection>` element, a SOAP security token request response (that is, a `<S:Envelope>`) as detailed in WS-Trust, or a SOAP `<S:Fault>` element. This carries the same content as a WS-Trust RSTR element (or even the actual SOAP Envelope containing the RSTR element).

wctx
: This optional parameter specifies the context information (if any) passed in with the request and typically represents context from the original request.

In the event that the token/result cannot be passed in the form, the following parameter MAY be specified:

```
wresultptr=url
```

wresultptr

This parameter specifies a URL to which an HTTP GET can be issued. The result is a document of type `text/xml` that contains the issuance result. This can either be the `<wst:RequestSecurityTokenResponse>` element, the `<wst:RequestSecurityTokenResponseCollection>` element, a SOAP response, or a SOAP `<S:Fault>` element. Note that this serves roughly the same purpose as the WS-ReferenceToken mechanism previously discussed (although this is used for the full response not just the token).

13.2.4. Sign-Out Request Syntax

This section describes how sign-out requests are formed and redirected by Web requestors. For modularity, it should be noted that support for sign-out is optional.

The following describes the parameters used for the sign-out request (note that this parallels the sign-out SOAP message previously discussed):

```
wa=string
wreply=URL
```

wa

This required parameter specifies the action to be performed. By including the action, URIs can be overloaded to perform multiple functions. For sign-out, this string MUST be "wsignout1.0". For backward compatibility with previous versions, this parameter MAY be “wsignoutcleanup1.0” but use of this is deprecated. That is, processors SHOULD use “wsignout1.0” but MUST accept “wsignoutcleanup1.0”.

wreply

This optional parameter specifies the URL to return to once clean-up (sign-out) is complete. If this parameter is not specified, then after cleanup the GET completes by returning any realm-specific data such as a string indicating cleanup is complete for the realm.

13.2.5. Attribute Request Syntax

This section describes how attribute requests are formed and redirected by Web requestors. For modularity, it should be noted that support for attributes is optional. Additionally it should be noted that security considerations may apply. While the structure described here can be used with any attribute service supporting Web clients, the actual attribute request and response XML syntax is undefined and specific to the attribute store.

The following describes the valid parameters used within attributes requests:

```
wa=string
[wreply=URL]
[wtrealm=URL]
wattr=xml-attribute-request
wattrptr=URL
wresult=xml-result
wresultptr-URL
```

wa

This required parameter specifies the action to be performed. By including the action, URIs can be overloaded to perform multiple functions. For attribute requests, this string MUST be "wattr1.0".

wreply
: This optional parameter specifies the URL to return to when the attribute result is complete.

wattr
: This optional parameter specifies the attribute request. The syntax is specific to the attribute store being used and is not mandated by this specification. This attribute is only present on the request.

wattrptr
: This optional parameter specifies URL where the request can be obtained.

wresult
: This optional parameter specifies the result as defined by the attribute store and is not mandated by this specification. This attribute is only present on the responses.

wresultptr
: This optional parameter specifies URL where the result can be obtained.

13.2.6. Pseudonym Request Syntax

This section describes how pseudonym requests are formed and redirected by Web requestors. For modularity, it should be noted that support for pseudonyms is also optional. As well, it should be noted that security considerations may apply.

The following describes the valid parameters used within pseudonym requests (note that this parallels the pseudonym messages previously discussed):

```
wa=string
[wreply=URL]
[wtrealm=URL]
wpseudo=xml-pseudonym-request
wpseudoptr=URL
wresult=xml-result
wresultptr=URL
```

wa
: This required parameter specifies the action to be performed. By including the action, URIs can be overloaded to perform multiple functions. For pseudonym requests, this string MUST be "wpseudo1.0".

wreply
: This optional parameter specifies the URL to return to when the pseudonym result is complete.

wpseudo
: This optional parameter specifies the pseudonym request and either contains a SOAP envelope or a pseudonym request, such as a WS-Transfer/WS-ResourceTransfer `<Get>`. This attribute is only present on the request.

wpseudoptr
: This optional parameter specifies URL from which the request element can be obtained.

wresult

This optional parameter specifies the result as either a SOAP envelope or a pseudonym response. This attribute is only present on the responses.

wresultptr

This optional parameter specifies URL from which the result element can be obtained.

13.3. Detailed Example of Web Requester Syntax

This section provides a detailed example of the protocol defined in this specification. The exact flow for Web sign-in scenarios can vary significantly; however, the following diagram and description depict a *common* or basic sequence of events.

In this scenario, the user at a requestor browser is attempting to access a resource which requires security authentication to be validated by the resource's security token service. In this example there is a resource-specific IP/STS.

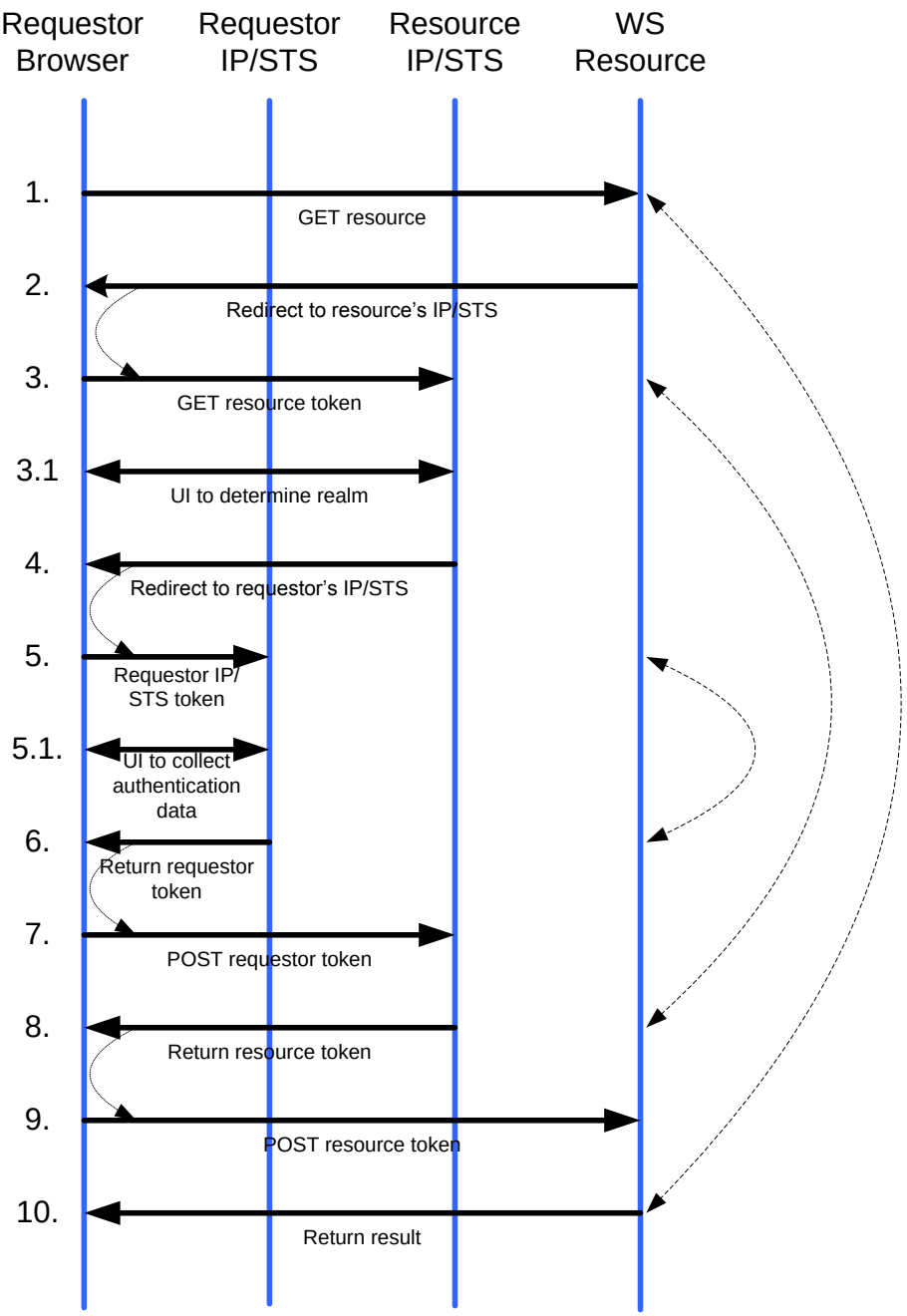


Figure 29: Details Sample Browser Sign-In

Simple Scenario:

This scenario depicts an initial federated flow. Note that subsequent flows from the requestor to the resource realm may be optimized. The steps below describe the above interaction diagram. Appendix III provides a set of sample HTTP messages for these steps.

Step 1: The requestor browser accesses a resource, typically using the HTTP GET method.

Step 2: At the resource, the requestor's request is redirected to the IP/STS associated with the target resource. The redirected URL MAY contain additional information reflecting agreements which the resource and its IP/STS have established; however, this (redirection

target) URL MUST be used throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method described in step 6).

It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of the information.

Step 3: Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor realm could be cached in an artifact/cookie from an earlier exchange, it could be known to or fixed by the resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

Step 3.1: This is an optional step. If the resource IP/STS cannot determine the requestor's realm, then the IP/STS MAY prompt the requestor for realm information.

Step 4: The resource IP/STS redirects to the requestor's IP/STS in order to validate the requestor. Typically, this is done using a HTTP 302 redirect.

As in step 2, additional information MAY be passed to reflect the agreement between the two IP/STS's, and this request for the token MAY be done using a POST method (see syntax for details).

The requestor IP/STS SHOULD provide information confidentiality or use HTTP/S or some other transport-level security mechanism.

Step 5: The requestor's IP/STS now authenticates the requestor to establish a sign in.

Step 5.1: Validation of the requestor MAY involve displaying some UI in this optional step.

Step 6: Once requestor information has been successfully validated, a security token response (RSTR) is formatted and sent to the resource IP/STS.

Processing continues at the resource IP/STS via a redirect.

While an IP/STS MAY choose to return a pointer to token information using wresultptr, it is RECOMMENDED that, whenever possible to return the security token (RSTR) using a POST method to reduce the number of overall messages. This MAY be done using requestor-side scripting. The exact syntax is described in Appendix I.

Step 7: Resource's IP/STS receives and validates the requestor's security token (RSTR).

Step 8: The resource's IP/STS performs a federated authentication/authorization check (validation against policy). After a successful check, the resource's IP/STS can issue a security token for the resource. The resource IP/STS redirects to the resource.

It should be noted that the optional `wctx` parameter specifies the opaque context information (if any) passed in with the original request and is echoed back here. This mechanism is an optional way for the IP/STS to have state returned to it.

At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the requestor (which likely includes the requestor's realm).

Step 9: The resource receives the security token (RSTR) from the resource IP/STS. On successful validation the resource processes the request (per policy).

The security token SHOULD be passed using an HTML POST using the syntax previously described.

Step 10: The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it returns the result of the resource request.

Optimized Scenario:

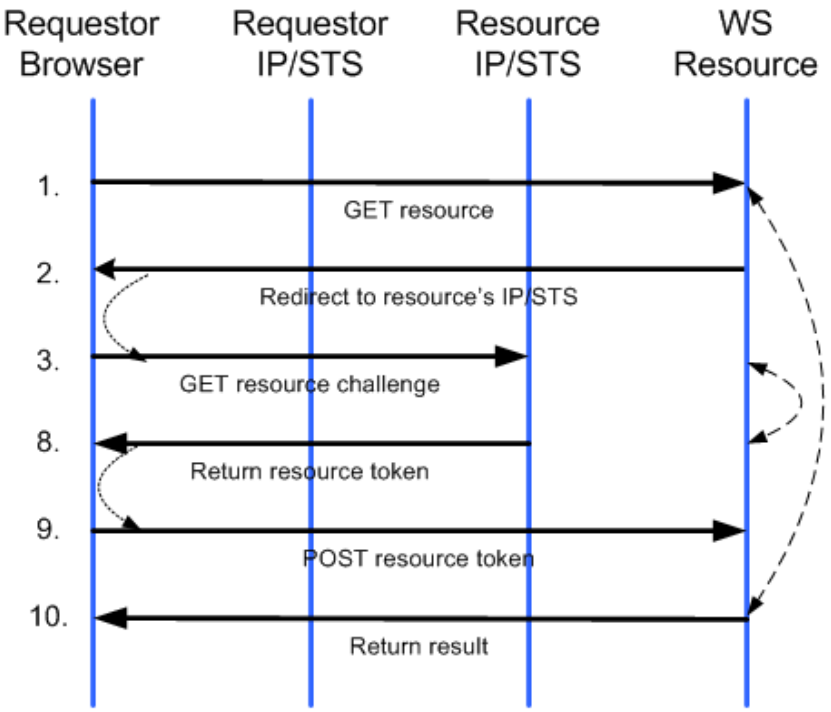


Figure 30: Optimized Sample Browser Sign-In

This scenario assumes that an initial federated flow has occurred. Note that many legs of the initial flow MAY be eliminated due to the presence of artifacts/cookies. For readability, the similar steps are numbered consistently with the previous non-optimized example.

Step 1: The requestor browser accesses a resource, typically using the HTTP GET method.

Step 2: At the resource, the requestor's request is redirected to the IP/STS associated with the target resource. The redirected URL MAY contain additional information reflecting agreements which the resource and its IP/STS have established; however, this (redirection target) URL MUST be used throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method described in step 6).

It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of the information.

Step 3: Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor realm could be cached in an artifact/cookie from an earlier exchange, it could be known to or fixed by the resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

Step 8: The resource's IP/STS performs a federated authentication/authorization check (validation against policy). After a successful check, the resource's IP/STS can issue a security token for the resource. The resource IP/STS redirects to the resource.

It should be noted that the optional `wctx` parameter specifies the opaque context information (if any) passed in with the original request and is echoed back here. This mechanism is an optional way for the IP/STS to have state returned to it.

At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the requestor (which likely includes the requestor's realm).

Step 9: The resource receives the security token (RSTR) from the resource IP/STS. On successful validation the resource processes the request (per policy).

The security token SHOULD be passed using an HTML POST using the syntax previously described.

Step 10: The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it returns the result of the resource request.

13.4. Request and Result References

The previous example illustrates a common form of messaging when passing WS-Trust messages via a simple Web browser. However, in some scenarios it is undesirable to use POST messages and carry the full details within the messages (e.g. when redirecting through wireless or mobile devices). In such cases requests and responses can be referenced via a URL and all messages passed as part of the query strings (or inside small POSTs).

Request references are specified via *wreqptr* and typically specify a `<wst:RequestSecurityToken>` element that can be obtained by issuing a HTTP GET against the specified URL. Response references are specified via *wresultptr* and typically specify a `<wst:RequestSecurityTokenResponse>` or `<wst:RequestSecurityTokenResponseCollection>` element that can be obtained by issuing a HTTP GET against the specified URL.

This section provides a detailed example of the use of references with the protocol defined in this specification. The exact flow for Web sign-in scenarios can vary significantly; however, the following diagram and description depict a *common* or basic sequence of events. Note that this example only illustrates result reference not request references and makes use of a resource-specific IP/STS.

In this scenario, the user at a requestor browser is attempting to access a resource which requires security authentication to be validated by the resource's security token service.

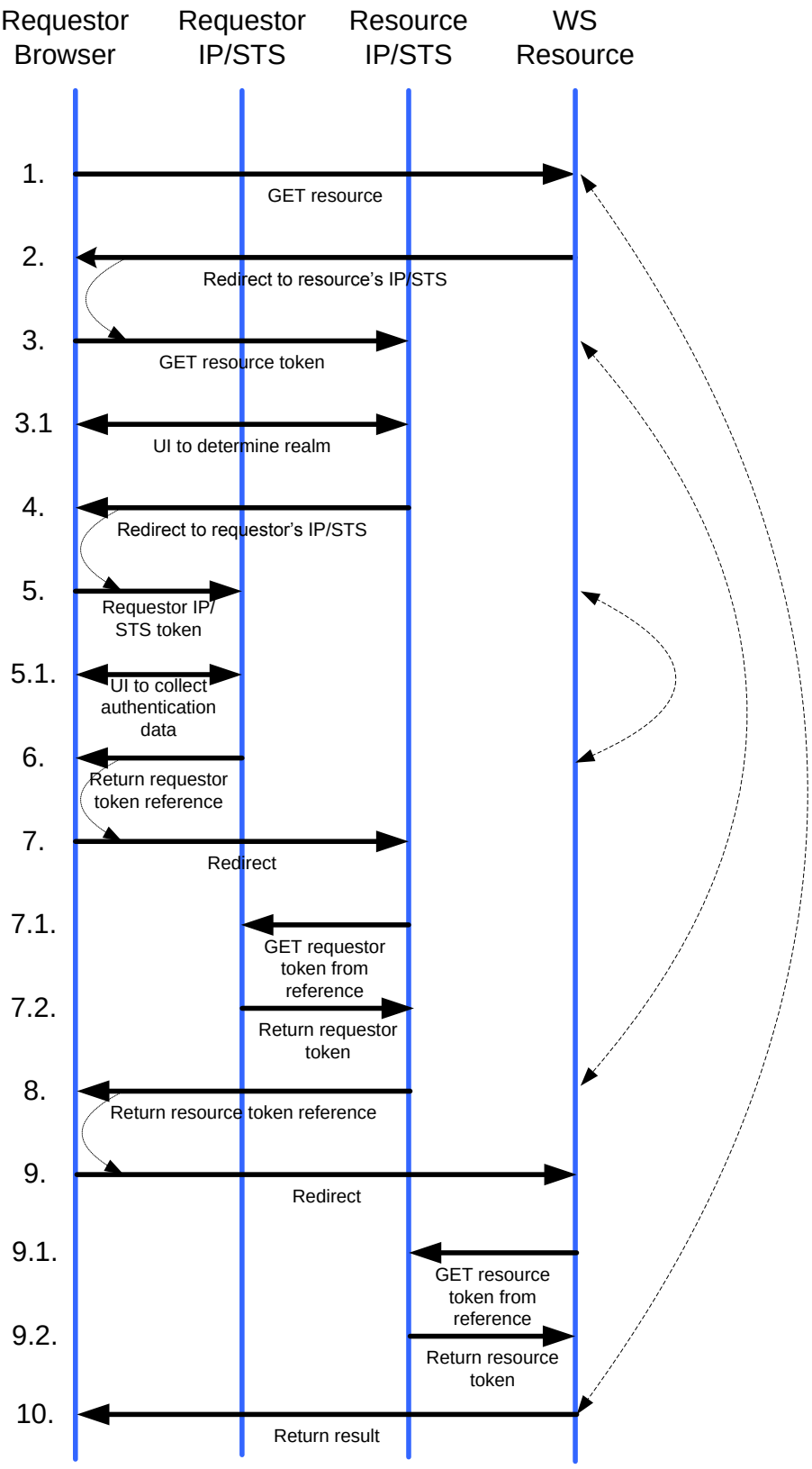


Figure 31: Sample Browser Sign-In with Request and Result References

Step 1: The requestor browser accesses a resource, typically using the HTTP GET method.

Step 2: At the resource, the requestor's request is redirected to the IP/STS associated with the target resource. The redirected URL MAY contain additional information reflecting agreements which the resource and its IP/STS have established; however, this (redirection target) URL MUST be used throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method described in step 6).

It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of the information.

Step 3: Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor realm could be cached in an artifact/cookie from an earlier exchange, it could be known to or fixed by the resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

Step 3.1: This is an optional step. If the resource IP/STS cannot determine the requestor's realm, then the IP/STS MAY prompt the requestor for realm information.

Step 4: The resource IP/STS redirects to the requestor's IP/STS in order to validate the requestor. Typically, this is done using a HTTP 302 redirect.

As in step 2, additional information MAY be passed to reflect the agreement between the two IP/STS's, and this request for the token MAY be done using a POST method (see syntax for details).

The requestor IP/STS SHOULD provide information confidentiality or use HTTP/S or some other transport-level security mechanism.

Step 5: The requestor's IP/STS now authenticates the requestor to establish a sign in.

Step 5.1: Validation of the requestor MAY involve displaying some UI in this optional step.

Step 6: Once requestor information has been successfully validated, a security token response (RSTR) is formatted and sent to the resource IP/STS.

Processing continues at the resource IP/STS via a redirect.

Step 7: Resource's IP/STS receives and validates the requestor's security token (RSTR).

Step 7.1: The Resource IP/STS issues a GET to the Requestor IP/STS to obtain the actual RSTR.

Step 7.2: The Requestor IP/STS responds to the GET and returns the actual RSTR.

Step 8: The resource's IP/STS performs a federated authentication/authorization check (validation against policy). After a successful check, the resource's IP/STS can issue a security token for the resource. The resource IP/STS redirects to the resource.

It should be noted that the optional `wctx` parameter specifies the opaque context information (if any) passed in with the original request and is echoed back here. This mechanism is an optional way for the IP/STS to have state returned to it.

At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the requestor (which likely includes the requestor's realm).

Step 9: The resource receives the security token (RSTR) from the resource IP/STS. On successful validation the resource processes the request (per policy).

The security token SHOULD be passed using an HTML POST using the syntax previously described.

Step 9.1: The Resource issues a GET to the Resource IP/STS to obtain the actual RSTR.

Step 9.2: The Resource IP/STS responds to the GET and returns the actual RSTR.

Step 10: The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it returns the result of the resource request.

13.5. Home Realm Discovery

In the protocol previously described the resource or the resource's IP/STS must determine the IP/STS for the requestor and re-direct to obtain an identity token. After this is done, the information can be cached in a cookie (or by whatever means is desired).

There is no normative way of discovering the *home realm* of the requestor, however, the following mechanisms are common methods:

- *Fixed* – The home realm is fixed or known
- *Requestor IP* – The home realm is determined using the requestor's IP address
- *Prompt* – The user is prompted (typically using a Web page)
- *Discovery Service* – A service is used to determine the home realm
- *Shared Cookie* – A shared cookie from a shared domain is used (out of scope)

The first three mechanisms are well understood, the *Discovery Service* is discussed next, and the cookie mechanism is outside the scope of this document.

13.5.1 Discovery Service

The *Home Realm Discovery Service* is a Web-based service that, through implementation-specific methods may be able to determine a requestor's home realm without user interaction.

A resource or resource IP/STS MAY redirect to a discovery service to attempt to determine the home realm without prompting the user. The discovery service MUST redirect back to the URL specified by the *wreply* parameter. If the context parameter is specified it MUST also be specified. If the discovery service was able to determine the home realm, it is returned using the *whr* parameter defined in section 13.2.2. This parameter contains a URI which identifies the home realm of the user. This SHOULD be the same URI that the user's realm uses for the wtrealm parameter when it makes token requests to other federated partners. This value can be used to lookup the URL for the user's IP/STS for properly redirecting the token request.

If the discovery service is unable to determine the home realm then the *whr* parameter is not specified and the home realm must be discovered by other means.

13.6. Minimum Requirements

For the purposes of interoperability of federated Web Single Sign-on, this sub-section defines a subset of the exchanges defined in this chapter which MUST be supported by all Web-enabled requestors and services. Optional aspects are optional for both clients and services.

The scenario and diagram(s) in section 13.3 illustrates the core Sign-On messages between two federated realms. This is the center of the interoperability subset described below.

13.6.1 Requesting Security Tokens

The focus of these requirements is on the message exchange between the requestor IP/STS and the resource IP/STS. Thus, to conform to this specification, messages 1, 4, 7 & 10 MUST be supported (again refer to the figure and steps in section 13.3). All other message exchanges are implementation specific and are only provided here for guidance.

A security token is requested via SignIn message in step 2 of the diagram. Message 3 arrives via HTTP GET and is protected by SSL/TLS. The parameters are encoded in a query string as specified in section 13.2. The message will contain parameters as detailed below. Parameters enclosed in brackets are OPTIONAL.

```
wa=wsignin1.0
wtrealm=resource realm URI
[wreply=Resource IP/STS Url]
[wctx=anything]
[wct=ISO8601 UTC]
```

The REQUIRED *wa* field is common to all SignIn messages and is fixed.

The REQUIRED *wtrealm* field MUST contain a URI that the *Resource IP/STS* and *Requestor IP/STS* have agreed to use to identify the realm of *Resource IP/STS* in messages to *Requestor IP/STS*.

The OPTIONAL *wreply* field specifies the URL to which this message's response will be POSTed (see Returning Security Tokens).

The OPTIONAL *wctx* field is provided for *Resource IP/STS*'s use and MUST be returned by *Requestor IP/STS* unchanged.

The OPTIONAL *wct* field, if present, MUST contain the current time in UTC using the ISO8601 format (e.g. "2003-04-30T22:47:20Z"). This field MAY not be available if the requestor is coming via a portal link. Individual implementations of *Requestor IP/STS* MAY require this field to be present.

Other options MAY be specified but are not REQUIRED to be supported.

13.6.2 Returning Security Tokens

A security token is returned in response to successful Web SignIn messages, as described in the example protocol message flow in section 13.3. Security tokens are returned to the requestor and SHOULD be transmitted to a Resource Provider via HTTP POST and be protected by SSL/TLS, as depicted in steps 6-7 and 9-10 of figure 29. Optionally, the token MAY be returned using the *wresultptr* parameter. Encoding of the parameters in the POST body MUST be supported. The parameters to the message MAY be encoded in the query string if *wresultptr* is being used. The message will contain parameters as detailed below. Parameters enclosed in brackets are OPTIONAL.

```
wa=wsignin1.0
wresult=RequestSecurityTokenResponse
[wctx=wctx from the request]
[wresultptr=URL]
```

The REQUIRED *wa* field is common to all SignIn messages and is fixed.

The REQUIRED *wresult* field MUST contain a <`wst:RequestSecurityTokenResponse`> element, as detailed below.

The OPTIONAL *wctx* field MUST be identical to the *wctx* field from the incoming SignIn message that evoked this response.

The OPTIONAL *wresultptr* field provides a pointer to the resulting `<wst:RequestSecurityTokenResponse>` element, as detailed below.

13.6.3 Details of the RequestSecurityTokenResponse element

The `<wst:RequestSecurityTokenResponse>` element that is included as the *wresult* field in the SignIn response MUST contain a `<wst:RequestedSecurityToken>` element. Support for SAML assertions MUST be provided but another token format MAY be used (depending on policy).

The `<wst:RequestSecurityTokenResponse>` element MAY include a *wsp:AppliesTo* / *wsa:EndpointReference* / *wsa:Address* element that specifies the Resource Realm URI. Note that this data MUST be consistent with similar data present in security tokens (if any is present) – for example it must duplicate the information in the signed token's *saml:Audience* element when SAML security tokens are returned.

13.6.4 Details of the Returned Security Token Signature

It MUST be possible to return signed security tokens, but unsecured tokens MAY be returned. Signed security tokens SHOULD contain an enveloped signature to prevent tampering but MAY use alternative methods if the security token format allows for specialized augmentation of the token. The signature SHOULD be performed over canonicalized XML [XML-C14N] (failure to do so may result in non-verifiable security tokens). The signature SHOULD be produced using the *Requestor STS* private key, which SHOULD correspond to either a security token included as part of the response or pre-established with the requestor. Note that in the above example the certificate is included directly in KeyInfo (via the X509Data element [WSS:X509Token]). This is the RECOMMENDED approach.

When used, the X509SKI element contains the base64 encoded plain (i.e., non-DER-encoded) value of an X509 V.3 SubjectKeyIdentifier extension. If the SubjectKeyIdentifier field is not present in the certificate, the certificate itself must be included directly in KeyInfo (see the above example).

Note that typically the returned security token is unencrypted (The entire RSTR is sent over SSL3.0/TLS [HTTPS]) but it MAY be encrypted in specialized scenarios.

Take care to include appropriate transforms in *Signature/Reference/Transforms*. For example, all SAML tokens [WSS:SAMLTokenProfile] following the rules above must contain the enveloped signature and EXCLUSIVE cannonicalization transforms.

13.6.5 Request and Response References

If the *wreqptr* or *wresultptr* parameters are supported, it MUST be possible to pass `<wst:RequestSecurityToken>` in the *wreqptr* and either `<wst:RequestSecurityTokenResponse>` or `<wst:RequestSecurityTokenResponseCollection>` in *wresultptr*. Other values MAY be supported but are NOT REQUIRED to be supported.

14. Additional Policy Assertions

This specification defines the following assertions for use with [WS-Policy] and [WS-SecurityPolicy].

14.1. RequireReferenceToken Assertion

This element represents a requirement to include a ReferenceToken (as described previously in this specification). The default version of this token is the version described in this document.

The syntax is as follows:

```
<fed:RequireReferenceToken sp:IncludeToken="xs:anyURI" ? ... >
  <wsp:Policy>
    <fed:RequireReferenceToken11 ...>...</fed:RequireReferenceToken11> ?
    ...
  </wsp:Policy> ?
  ...
</fed:RequireReferenceToken>
```

The following describes the attributes and elements listed in the schema outlined above:

/fed:RequireReferenceToken

This identifies a RequireReference assertion

/fed:RequireReferenceToken/sp:IncludeToken

This optional attribute identifies the token inclusion value for this token assertion

/fed:RequireReferenceToken/wsp:Policy

This optional element identifies additional requirements for use of the fed:RequireReferenceToken assertion.

/fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11

This optional element indicates that a reference token should be used as defined in this specification.

/fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11/@{any}

This extensibility mechanism allows attributes to be added so long as they don't violate or alter the semantics defined in this specification.

/fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11/{any}

This is an extensibility point allowing content elements to be specified. Note that no element can alter semantic defined in this specification.

/fed:RequireReferenceToken/@{any}

This extensibility mechanism allows attributes to be added so long as they don't violate or alter the semantics defined in this specification.

/fed:RequireReferenceToken/{any}

This is an extensibility point allowing content elements to be specified. Note that no element can alter semantic defined in this specification.

This assertion is used wherever acceptable token types are identified (e.g. within the supporting token assertions defined in WS-SecurityPolicy).

14.2. WebBinding Assertion

The WebBinding assertion is used in scenarios where requests are made of token services using a Web client and HTTP with GET, POST, and redirection as described in Section 13. Specifically, this assertion indicates that the requests use the Web client mechanism defined

in this document and are protected using the means provided by a transport. This binding has several specific binding properties:

- The [TransportToken] property indicates what transport mechanism is used to protect requests and responses.
- The [AuthenticationToken] property indicates the required token type for authentication. Note that this can be a choice of formats as it uses nested policy. Also note that this can specify fed:ReferenceToken as an option to indicate that token handles are accepted (and dereferenced).
- The [RequireSignedTokens] property indicates that only tokens that are signed are accepted.
- The [RequireBearerTokens] property indicates that only bearer tokens are accepted.
- The [RequireSharedCookies] property indicates if shared cookies must be used for home realm discovery

The syntax is as follows:

```
<fed:WebBinding ...>
  <wsp:Policy>
    <sp:TransportToken ...> ... </sp:TransportToken> ?
    <fed:AuthenticationToken ... > ?
      <wsp:Policy> ... </wsp:Policy>
      <fed:ReferenceToken ...>... </fed:ReferenceToken> ?
    </fed:AuthenticationToken>    <fed:RequireSignedTokens ... /> ?
    <fed:RequireBearerTokens ... /> ?
    <fed:RequireSharedCookies ... /> ?
    ...
  </wsp:Policy> ?
</fed:WebBinding>
```

The following describes the attributes and elements listed in the schema outlined above:

/fed:WebBinding

This identifies a WebBinding assertion

/fed:WebBinding/wsp:Policy

This identifies a nested `wsp:Policy` element that defines the behavior of the WebBinding assertion.

/fed:WebBinding/wsp:Policy/sp:TransportToken

This indicates a requirement for a Transport Token as defined in [WS-SecurityPolicy].

/fed:WebBinding/wsp:Policy/fed:AuthenticationToken

This indicates the required token type for authentication.

/fed:WebBinding/wsp:Policy/fed:AuthenticationToken/wsp:Policy

This indicates a nested wsp:Policy element to specify a choice of formats for the authentication token.

/fed:WebBinding/wsp:Policy/fed:AuthenticationToken/fed:ReferenceToken

This optional element indicates token handles that are accepted. See section 8.1 for a complete description.

/fed:WebBinding/wsp:Policy/RequireSignedTokens

This indicates a requirement for tokens to be signed. This sets the [RequireSignedTokens] property to true (the default value is false).

/fed:WebBinding/wsp:Policy/RequireBearerTokens

This indicates a requirement for bearer tokens. This sets the [RequireBearerTokens] property to *true* (the default value is *false*).

/fed:WebBinding/wsp:Policy/RequireSharedCookies

This indicates a requirement for shared cookies to facilitate home realm discovery. This sets the [RequireSharedCookies] property to *true* (the default value is *false*).

Note that the `sp:AlgorithmSuite`, `sp:Layout`, and `sp:IncludeTimestamp` properties are not used by this binding and SHOULD NOT be specified.

This assertion SHOULD only be used with endpoint subjects.

14.3. Authorization Policy

To indicate support for the authorization features described in this specification, the following policy assertions are specified.

```
<fed:RequiresGenericClaimDialect ... />
<fed:IssuesSpecificMetadataFault ... />
<fed:AdditionalContextProcessed ... />
```

The following describes the above syntax:

/fed:RequiresGenericClaimDialect

This assertion indicates that the service requires the use of the generic claim dialect defined in this specification in Section 9.3.

/fed:IssuesSpecificPolicyFault

This assertion indicates that the service issues the `fed:SpecificPolicy` Fault defined in this document if the security requirements for a specific request are beyond those of the base policy.

/fed:AdditionalContextProcessed

This assertion indicates that the service will process the `fed:AdditionalContext` parameter if specified in an RST request.

Typically these assertions are specified at the service or port/endpoint.

These assertions SHOULD be specified within a binding assertion.

15 Error Handling

This specification defines the following error codes that MAY be used. Other errors MAY also be used. These errors use the SOAP Fault mechanism. Note that the reason text provided below is RECOMMENDED, buy alternative text MAY be provided if more descriptive or preferred by the implementation. The table below is defined in terms of SOAP 1.1. For SOAP 1.2 the Fault/Code/Value is env:Sender (as defined in SOAP 1.2) and the Fault/Code/SubCode/Value is the *faultcode* below, and the Fault/Reason/Text is the *faultstring* below. It should be notes that profiles MAY provide second-level detail fields but they should be careful not to introduce security vulnerabilities when doing so (e.g. by

providing too detailed information or echoing confidential information over insecure channels). It is RECOMMENDED that Faults use the indicated action URI when sending the Fault.

Error that occurred (faultstring)	Fault code (faultcode)	Fault Action URI
No pseudonym found for the specified scope	`fed:NoPseudonymInScope`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/NoPseudonymInScope
The principal is already signed in (optional – need not be reported)	`fed:AlreadySignedIn`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/AlreadySignedIn
The principal is not signed in (optional – need not be reported)	`fed:NotSignedIn`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/NotSignedIn
An improper request was made (e.g., Invalid/unauthorized pseudonym request)	`fed:BadRequest`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/BadRequest
No match for the specified scope	`fed:NoMatchInScope`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/NoMatchInScope
Credentials provided don't meet the freshness requirements	`fed:NeedFresherCredentials`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/NeedFresherCredentials
Specific policy applies to the request – the new policy is specified in the `S12:Detail` element.	`fed:SpecificPolicy`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/SpecificPolicy
The specified dialect for claims is not supported	`fed:UnsupportedClaimsDialect`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/UnsupportedClaimsDialect

Error that occurred (faultstring)	Fault code (faultcode)	Fault Action URI
A requested RST parameter was not accepted by the STS. The details element contains a `fed:Unaccepted` element. This element's value is a list of the unaccepted parameters specified as QNames.	`fed:RstParameterNotAccepted`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/RstParameterNotAccepted
A desired issuer name is not supported by the STS	`fed:IssuerNameNotSupported`	http://schemas.xmlsoap.org/ws/2006/12/federation/Fault/IssuerNameNotSupported

16. 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.

Metadata that is exchanged also needs to be secured to prevent various attacks. All metadata documents should be verified to ensure that the issuer can speak for the specified endpoint and that the metadata is what the issuer intended.

All federation-related messages such as sign-out, principal, attribute, and pseudonym management should be integrity protected (signed or use transport security). If a message is received where the body is not integrity protected, 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.

When using Web messages care should be taken around processing of the *wreply* parameter as its value could be spoofed. It is recommended that implementations do explicit lookup and verification of URL, and that these values be passed with transport security.

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 to protect itself), it should be included in any message integrity mechanisms (e.g. included in the message signature).

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

Care should be taken when processing and responding to requests from 3rd-parties to mitigate potential information disclosure attacks by way of faulting requests for specific claims.

As a general rule tokens should not have lifetimes beyond the minimum of the basis credentials (security tokens). However, in some cases special arrangements may exist and issuers may provide longer lived tokens. Care should be taken in such cases not to introduce security vulnerabilities.

The following list summarizes common classes of attacks that apply to this protocol and identifies the mechanism to prevent/mitigate the attacks. Note that wherever WS-Security is suggested as the mitigation, [HTTPS] is the corresponding mechanism for Web requestors:

- **Metadata alteration** – Alteration is prevented by including signatures in metadata or using secure channels for metadata transfer.
- **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 recommended 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.

17. Acknowledgements

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

John Favazza, CA
Tim Hahn, IBM
Andrew Hatley, IBM
Heather Hinton, IBM
Michael McIntosh, IBM
Anthony Moran, IBM
Birgit Pfitzmann, IBM
Bruce Rich, IBM
Shane Weeden, IBM
Jan Alexander, Microsoft
Greg Carpenter, Microsoft
Paul Cotton, Microsoft
Marc Goodner, Microsoft
Martin Gudgin, Microsoft
Savas Parastatidis, Microsoft

18. References

[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.
http://ietf.org/rfc/rfc2616.txt

[HTTPS]
IETF Standard, "The TLS Protocol", January 1999.
http://www.ietf.org/rfc/rfc2246.txt

[KEYWORDS]
S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.
http://www.ietf.org/rfc/rfc2119.txt.

[SOAP]
W3C Note, "SOAP: Simple Object Access Protocol 1.1", 08 May 2000.
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SOAP12]
W3C Recommendation, "SOAP 1.2 Part 1: Messaging Framework", 24 June 2003.

http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[URI]
T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 3986, MIT/LCS, Day Software, Adobe Systems, January 2005.
http://www.ietf.org/rfc/rfc3986.txt

[WS-Addressing]
W3C Recommendation, "Web Services Addressing (WS-Addressing)", 9 May 2006.
http://www.w3.org/TR/2006/REC-ws-addr-core-20060509

[WS-Eventing]
W3C Member Submission, "Web Services Eventing (WS-Eventing)", 15 March 2006
http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/

[WS-MetadataExchange]
Web Services Metadata Exchange (WS-MetadataExchange), August 2006
http://schemas.xmlsoap.org/ws/2004/09/mex/

[WS-Policy]
W3C Member Submission "Web Services Policy 1.2 - Framework", 25 April 2006.
http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/

[WS-PolicyAttachment]
W3C Member Submission "Web Services Policy 1.2 - Attachment", 25 April 2006.
http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/

[WS-SecureConversation]
OASIS Committee Draft, "WS-SecureConversation 1.3", September 2006
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512

[WS-SecurityPolicy]
OASIS Committee Draft, "WS-SecurityPolicy 1.3", September 2006
http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512

[WS-Security]
OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", March 2004.
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[WS-Security11]
OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", February 2006.
http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[WSS:UsernameToken]
OASIS Standard, "Web Services Security: UsernameToken Profile", March 2004
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

OASIS Standard, "Web Services Security: UsernameToken Profile 1.1", February 2006

http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf

[WSS:X509Token]

OASIS Standard, "Web Services Security X.509 Certificate Token Profile", March 2004

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

OASIS Standard, "Web Services Security X.509 Certificate Token Profile", February 2006

http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf

[WSS:KerberosToken]

OASIS Standard, "Web Services Security Kerberos Token Profile 1.1", February 2006

http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf

[WSS:SAMLTokenProfile]

OASIS Standard, "Web Services Security: SAML Token Profile", December 2004

http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf

OASIS Standard, "Web Services Security: SAML Token Profile 1.1", February 2006

http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf

[WS-ResourceTransfer]

W3C Member Submission, "Web Services Resource Transfer (WS-ResourceTransfer)", August 2006

http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/

[WS-Transfer]

Web Services Transfer (WS-Transfer), 27 September 2006
http://www.w3.org/Submission/2006/SUBM-WS-Transfer-20060927/

[WS-Trust]

OASIS Committee Draft, "WS-Trust 1.3", September 2006

http://docs.oasis-open.org/ws-sx/ws-trust/200512

[ISO8601]

ISO Standard 8601:2004(E), "Data elements and interchange formats – Information interchange - Representation of dates and times", Third edition, December 2004

http://isotc.iso.org/livelink/livelink/4021199/ISO_8601_2004_E.zip?func=doc.Fetch&nodeid=4021199

[DNS-SRV-RR]

Gulbrandsen, et al, RFC 2782, "DNS SRV RR", February 2000.

http://ietf.org/rfc/rfc2782.txt

[XML-Infoset]

W3C Recommendation, "XML Information Set (Second Edition)", 4 February 2004

http://www.w3.org/TR/2004/REC-xml-infoset-20040204/

[XML-Schema1]

W3C Recommendation, "XML Schema Part 1: Structures Second Edition", 28 October 2004.

http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[XML-Schema2]
W3C Recommendation, "XML Schema Part 2: Datatypes Second Edition", 28 October 2004.

http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[XML-C14N]
W3C Recommendation, "Canonical XML Version 1.0", 15 March 2001

http://www.w3.org/TR/2001/REC-xml-c14n-20010315

[XML-Signature]
W3C Recommendation, "XML-Signature Syntax and Processing", 12 February 2002

http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[WSDL 1.1]
W3C Note, "Web Services Description Language (WSDL 1.1)," 15 March 2001.

http://www.w3.org/TR/2001/NOTE-wsdl-2001031

[XPATH]
W3C Recommendation "XML Path Language (XPath) Version 1.0", 16 November 1999.

http://www.w3.org/TR/1999/REC-xpath-19991116

Appendix I - WSDL

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

```
<wsdl:definitions xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
      xmlns:xs='http://www.w3.org/2001/XMLSchema'
      xmlns:tns='http://schemas.xmlsoap.org/ws/2006/12/federation'
      targetNamespace='http://schemas.xmlsoap.org/ws/2006/12/federation' >

<!-- WS-Federation endpoints implement WS-Trust -->
<wsdl:import namespace='http://docs.oasis-open.org/ws-sx/ws-trust/200512
location='http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3.wsdl'
/>

<!-- WS-Federation endpoints can implement WS-MEX -->
<wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/09/mex'
location='http://schemas.xmlsoap.org/ws/2004/09/mex/MetadataExchange.wsdl' />

<!-- WS-Federation endpoints can implement WS-Eventing -->
<wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/08/eventing'
location='http://schemas.xmlsoap.org/ws/2004/08/eventing/eventing.wsdl' />
```

```
<!-- WS-Federation endpoints can implement WS-Transfer -->
<wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/09/transfer'
location='http://schemas.xmlsoap.org/ws/2004/09/transfer/transfer.wsdl'/>

<!-- WS-Federation endpoints can implement WS-ResourceTransfer -->
<wsdl:import
namespace='http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer'
location='http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/wsrt.wsdl'
/>

<wsdl:types>
 <xs:schema>
  <xs:import namespace='http://schemas.xmlsoap.org/ws/2006/12/federation' />
 </xs:schema>
</wsdl:types>

<wsdl:message name='SignOut' >
 <wsdl:part name='Body' element='tns:SignOut' />
</wsdl:message>

<wsdl:portType name='SignOutIn' >
  <wsdl:operation name='SignOut' >
    <wsdl:input message='tns:SignOut' />
  </wsdl:operation>
</wsdl:portType>

<wsdl:portType name='SignOutOut' >
  <wsdl:operation name='SignOut' >
    <wsdl:output message='tns:SignOut' />
  </wsdl:operation>
</wsdl:portType>

</wsdl:definitions>
```

Appendix II. Sample HTTP Flows for Web Requestor Detailed Example

This appendix provides sample HTTP messages for the detailed example previously described in the Web requestor section.

In this example, the following URLs are used:

Item	***URL***
Resource Realm	Resource.com
Resource	https://res.resource.com/sales
Resource's IP/STS	https://sts.resource.com/sts
Account	Account.com
Resource	https://sts.account.com/sts

Step 1 – GET resource

```
GET https://res.resource.com/sales HTTP/1.1
```

Step 2 – Redirect to resource's IP/STS

```
HTTP/1.1 302 Found ↵
Location:
https://sts.resource.com/sts?wa=wsignin1.0&wreply=https://res.resource.com/sales&wct=2003-03-03T19:06:21Z
```

In addition, the resource could check for a previously written artifact/cookie and, if present, skip to Step 10.

Step 3 – GET resource challenge

```
GET https://sts.resource.com/sts?wa=wsignin1.0&wreply=
https://res.resource.com/sales&wct=2003-03-03T19:06:21Z HTTP/1.1
```

Step 3.1 – UI to determine realm (OPTIONAL)

```
 [Implementation Specific Traffic]
```

Step 4 – Redirect to requestor's IP/STS

```
HTTP/1.1 302 Found ↵
Location: https://sts.account.com/sts?wa=wsignin1.0&wreply=
https://sts.resource.com/sts&wctx= https://res.resource.com/sales&wct=2003-03-03T19:06:22Z&wtrealm=resource.com
```

In addition, the Resource IP/STS MAY check for a previously written artifact/cookie and, if present, skip to Step 8.

Step 5 – Requestor IP/STS challenge

```
GET
https://sts.account.com/sts?wa=wsignin1.0&wreply=https://sts.resource.com/sts&wctx=https://res.resource.com/sales&wct=2003-03-03T19:06:22Z&wtrealm=resource.com HTTP/1.1
```

Step 5.1 – UI to collect authentication data (OPTIONAL)

```
 [Implementation Specific Traffic]
```

Step 6 – Return requestor token

```
HTTP/1.1 200 OK
...

<html xmlns="https://www.w3.org/1999/xhtml">
<head>
<title>Working...</title>
</head>
<body>
<form method="post" action="https://sts.resource.com/sts">
<p>
<input type="hidden" name="wa" value="wsignin1.0" />
<input type="hidden" name="wctx" value="https://res.resource.com/sales" />
<input type="hidden" name="wresult"
value="&lt;RequestSecurityTokenResponse&gt;...&lt;/RequestSecurityTokenRespon
se&gt;" />
<button type="submit">POST</button> <!-- included for requestors that do not
support javascript -->
</p>
</form>
<script type="text/javascript">
setTimeout('document.forms[0].submit()', 0);
</script>
</body>
</html>
```

Step 7 – POST requestor token

```
POST https://sts.resource.com/sts HTTP/1.1 ↵
… ↵
↵
wa=wsignin1.0 ↵
wctx=https://res.resource.com/sales
wresult=<RequestSecurityTokenResponse>…</RequestSecurityTokenResponse>
```

Step 8 – Return resource token

```
HTTP/1.1 200 OK
…
```

```
<html xmlns="https://www.w3.org/1999/xhtml">
<head>
<title>Working...</title>
</head>
<body>
<form method="post" action="https://res.resource.com/sales">
<p>
<input type="hidden" name="wa" value="wsignin1.0" />
<input type="hidden" name="wresult"
value="&lt;RequestSecurityTokenResponse&gt;...&lt;/RequestSecurityTokenRespon
se&gt;" />
<button type="submit">POST</button> <!-- included for requestors that do not
support javascript -->
</p>
</form>
<script type="text/javascript">
setTimeout('document.forms[0].submit()', 0);
</script>
</body>
</html>
```

Step 9 – POST Resource token

```
POST https://res.resource.com/sales HTTP/1.1 ↵
... ↵
↵
wa=wsignin1.0 ↵
wresult=<RequestSecurityTokenResponse>...</RequestSecurityTokenResponse>
```

Step 10 – Return result

```
[Implementation Specific Traffic]
```

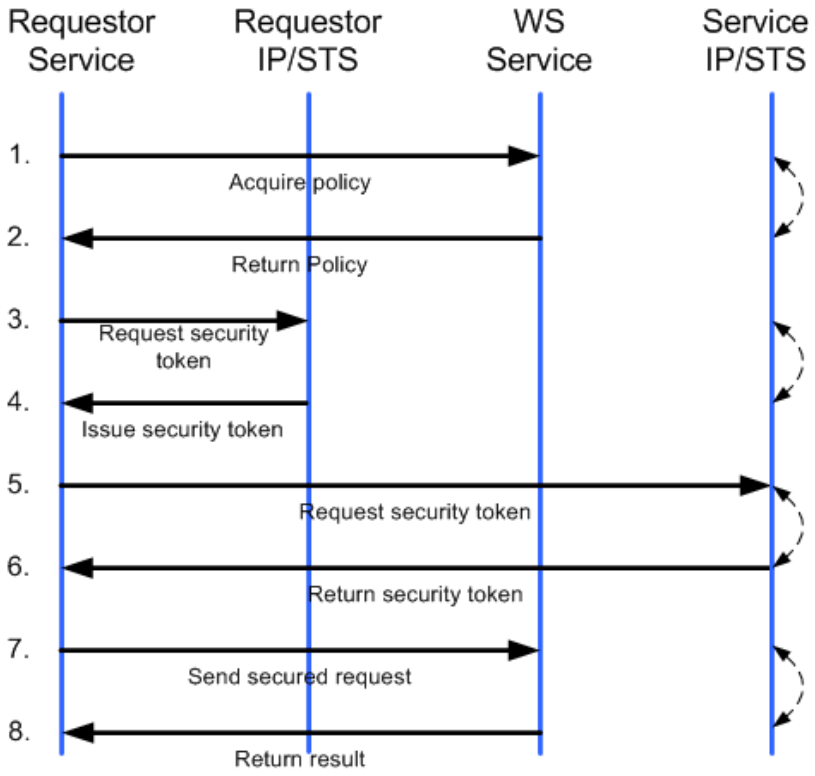
Appendix III. Sample Use Cases

The following sub-sections describe several use case scenarios and how they could be supported using this specification. Note that for each scenario there are potentially multiple ways to apply the messages and patterns in this specification so these examples should not be interpreted as the only or even the best approach, just an exemplary approach.

III.1. Single Sign On

Requestors use the mechanisms defined within [WS-Security], [WS-Trust], and [WS-Federation] to effect single sign-on.

At a high-level, policy is used to indicate communication requirements. Requestors can obtain the policy ahead of time or via error responses from services. In general, requestors are required to obtain a security token (or tokens) from their Identity Provider (or STS) when they authenticate themselves. The IP/STS generates a security token for use by the federated party. This is done using the mechanisms defined in WS-Trust. In some scenarios, the target service acts as its own IP/STS so communication with an additional service isn't required. Otherwise the requestor MAY be required to obtain additional security tokens from service-specific or service-required identity providers or security token services. The figure below illustrates one possible flow.



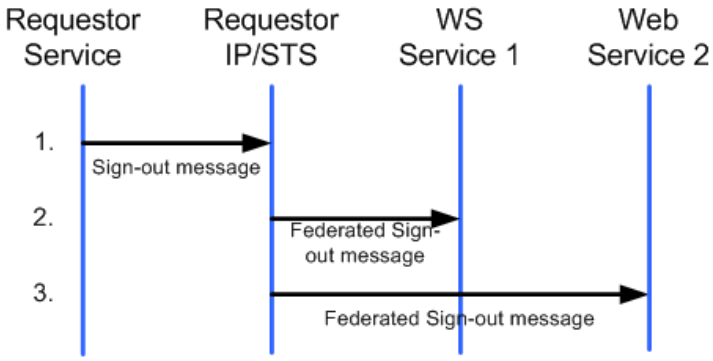
While the example above doesn't illustrate this, it is possible that the WS-Trust messages for security tokens MAY involve challenges to the requestors. Refer to WS-Trust for additional information.

III.2. Sign-Out

Just as it isn't typical for Web Service requestors to sign-in as a special operation, it isn't typical to *sign-out* either. However, for those scenarios where this is desirable, the sign-out messages defined in this specification can be used.

In situations where federated sign-out messages are desirable, the requestor's IP/STS SHOULD keep track of the realms to which it has issued tokens – specifically the IP/STS for the realms (or resources if different). When the sign-out is received at the requestor's IP/STS, the requestor's IP/STS is responsible for issuing federated sign-out messages to interested and authorized parties. The exact mechanism by which this occurs is up to the IP/STS, but it is strongly recommended that the sign-out messages defined in WS-Federation be used.

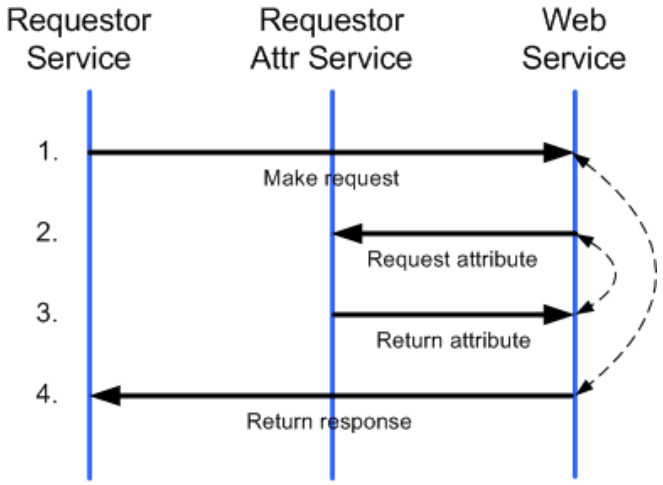
When a federated sign-out message is received at a realm, the realm should clean-up any cached information and delete any associated state as illustrated in the figure below:



III.3. Attributes

For Web Service requestors, attribute services are identified via WS-Policy or metadata as previously described. Web services and other authorized parties can obtain or even update attributes using the messages defined by the specific attribute service.

The figure below illustrates a scenario where a requestor issues a request to a Web service. The request MAY include the requestor's policy or it may be already cached at the service or the requestor MAY use [WS-MetadataExchange]. The Web service issues a request to the requestor's attribute service to obtain the values of a few attributes; WS-Policy MAY be used to describe the location of the attribute service. The service is authorized so the attributes are returned. The request is processed and a response is returned to the requestor.

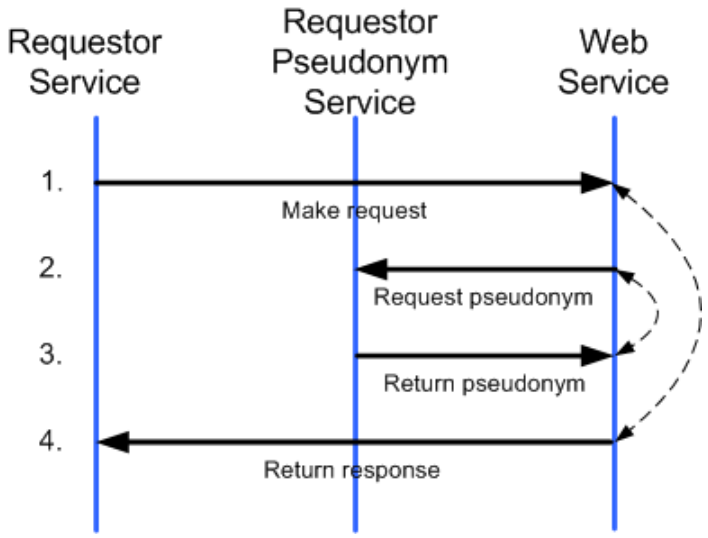


III.4. Pseudonyms

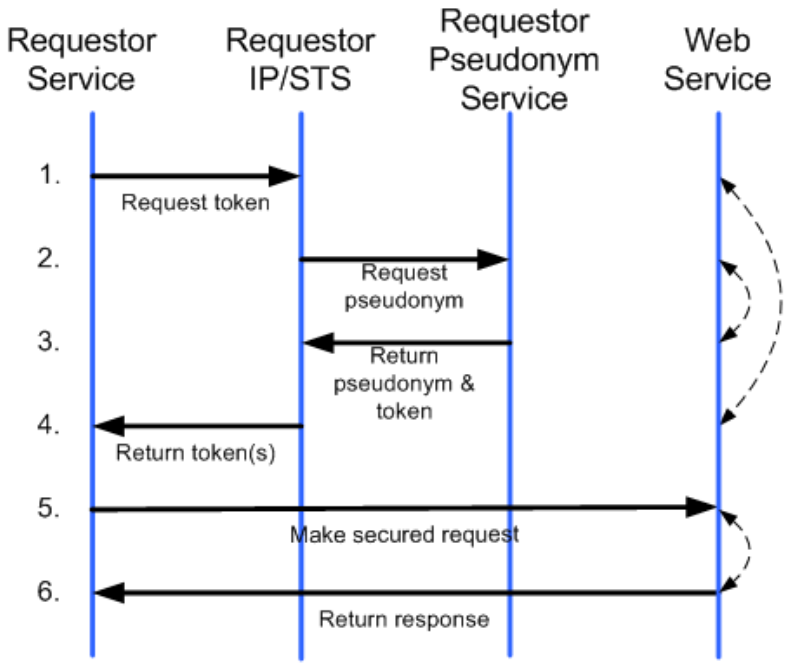
For Web Service requestors, pseudonym services are identified via metadata as previously described. Services and other authorized parties can obtain or manage pseudonyms using the messages previously defined.

The figure below illustrates a scenario where a requestor issues a request to a Web service. The request MAY include the requestor's policy and the location of the requestor’s pseudonym service or it may be already cached at the Web service. The Web service issues a request to the requestor's pseudonyms service to obtain the pseudonyms that are

authorized by the security token. The Web service is authorized so the pseudonym is returned. The request is processed and a response is returned to the requestor.



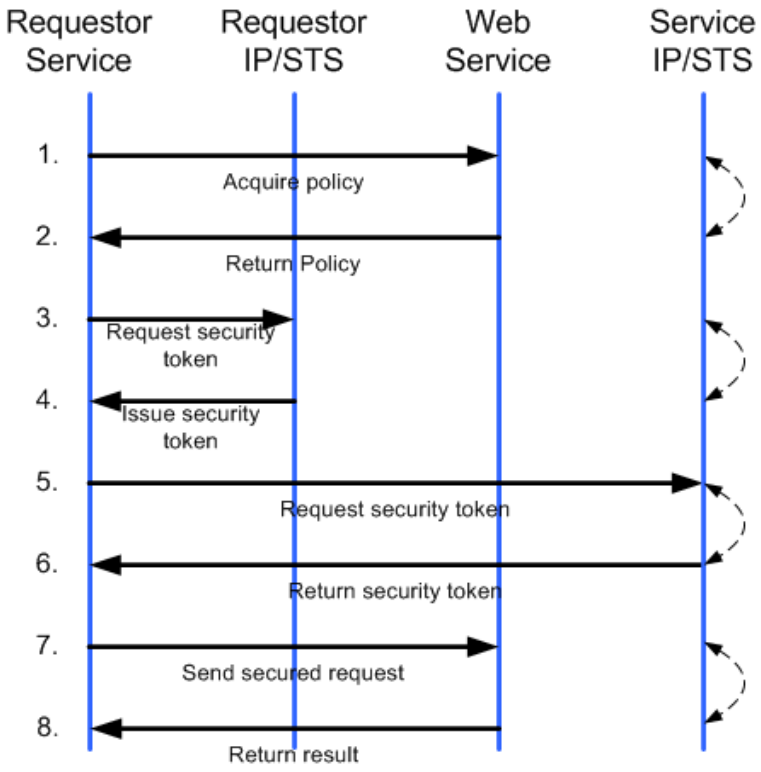
As previously described, the pseudonym and IP/STS can interact as part of the token issuance process. The figure below illustrates a scenario where a requestor has previously associated a pseudonym and a security token for a specific realm. When the requestor requests a security token to the domain/realm, the pseudonym and token are obtained and returned to the requestor. The requestor uses these security tokens for accessing the Web service.



III.5. Detailed Example

This section provides a detailed example of the protocol defined in this specification. The exact flow can vary significantly; however, the following diagram and description depict a *common* sequence of events.

In this scenario, a SOAP requestor is attempting to access a service which requires security authentication to be validated by the resource's security token service.



Step 1: Acquire Policy

If the requestor doesn't already have the policy for the service, it can obtain it using the mechanisms defined in WS-MetadataExchange.

Step 2: Return Policy

The requested policy is returned using the mechanisms defined in WS-MetadataExchange.

Step 3: Request Security Token

The requestor requests a security token from its IP/STS (assuming short-lived security tokens) using the mechanisms defined in WS-Trust (`<RequestSecurityToken>`)

Step 4: Issue Security Token

The IP/STS returns a security token (and optional proof of possession information) using the mechanisms defined in WS-Trust (`<RequestSecurityTokenResponse>` and `<RequestedProofToken>`)

Step 5: Request Security Token

The requestor requests a security token from the Web services IP/STS for the target Web service using the mechanisms defined in WS-Trust (`<RequestSecurityToken>`). Note that this is determined via policy or some out-of-band mechanism.

Step 6: Issue Security Token

The Web service's IP/STS returns a token (and optionally proof of possession information) using the mechanisms defined in WS-Trust (`<RequestSecurityTokenResponse>`)

Step 7: Send secured request

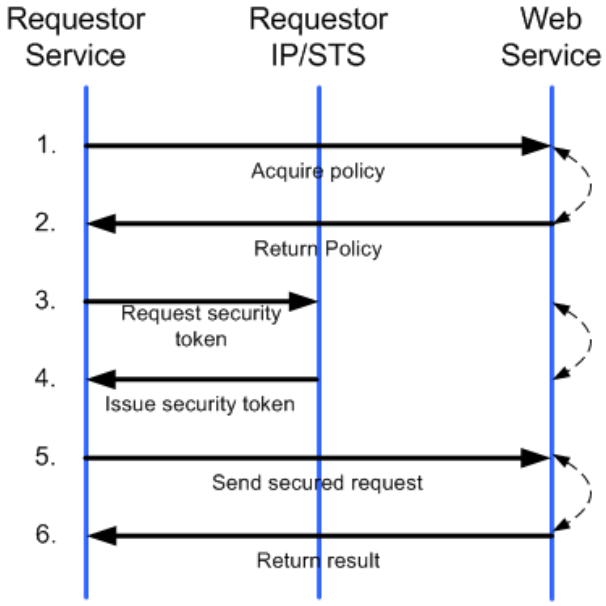
The requestor sends the request to the service attaching and securing the message using the issued tokens as described in WS-Security.

Step 8: Return result

The service issues a secured reply using its security token.

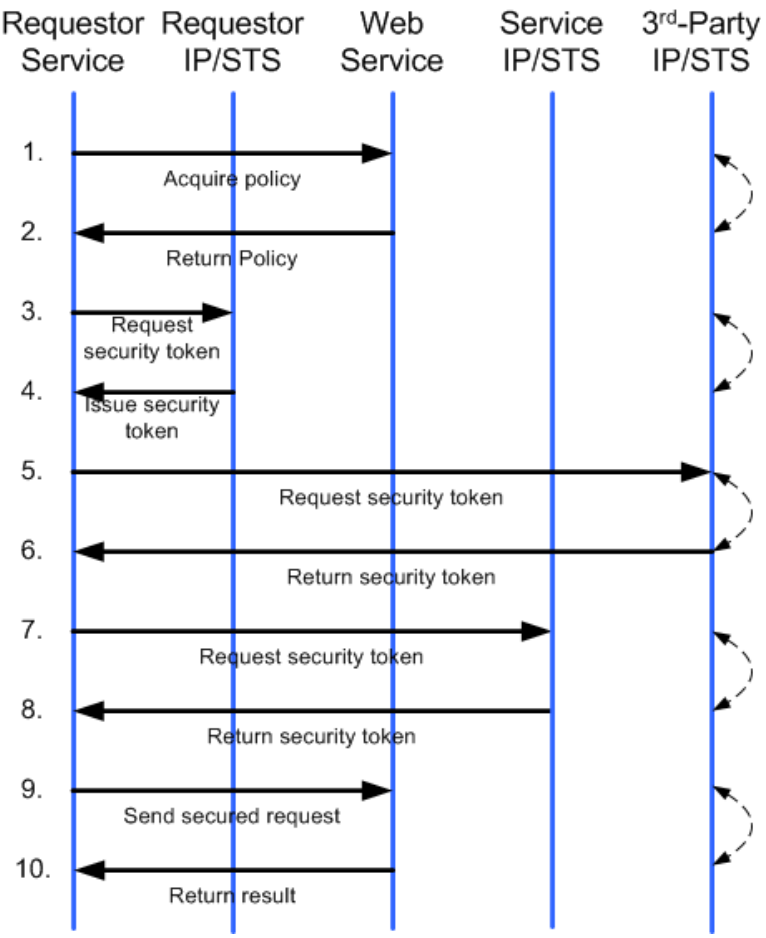
III.6. No Resource STS

The figure below illustrates the resource access scenario above, but without a resource STS. That is, the Web service acts as its own STS:



III.7. 3rd-Party STS

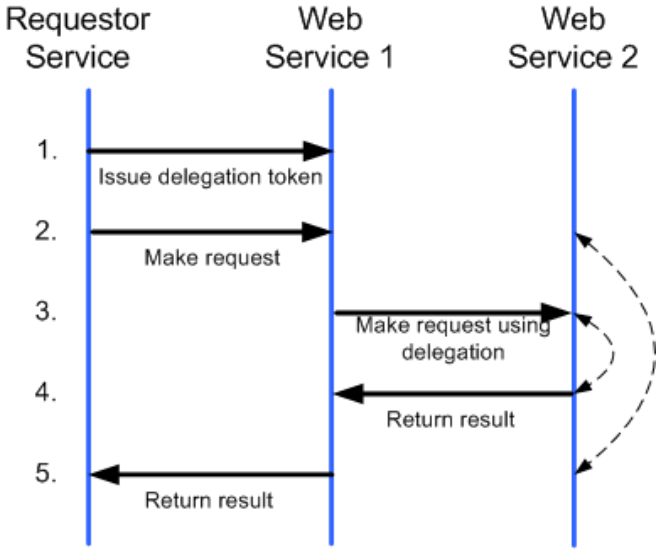
The figure below illustrates the resource access scenario above, but trust is brokered through a 3rd-party STS:



Note that 3rd-Party IP/STS is determined via policy or some out-of-band mechanism.

III.8. Delegated Resource Access

The figure below illustrates where a resource accesses data from another resource on behalf of the requestor:



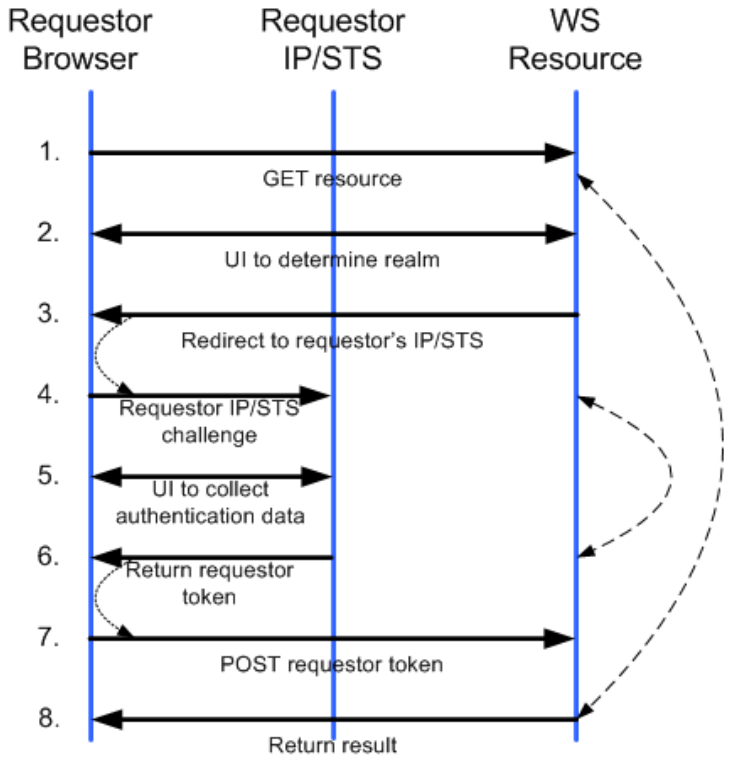
In this example, the requestor used a `<RequestSecurityTokenResponse>` as defined in WS-Trust to issue the delegation token in Step 1. This provides to Web Service 1 the necessary information so that Web Service 1 can act on the requestor's behalf as it contacts Web Service 2.

III.9. Additional Web Examples

This section presents interaction diagrams for additional Web requestor scenarios.

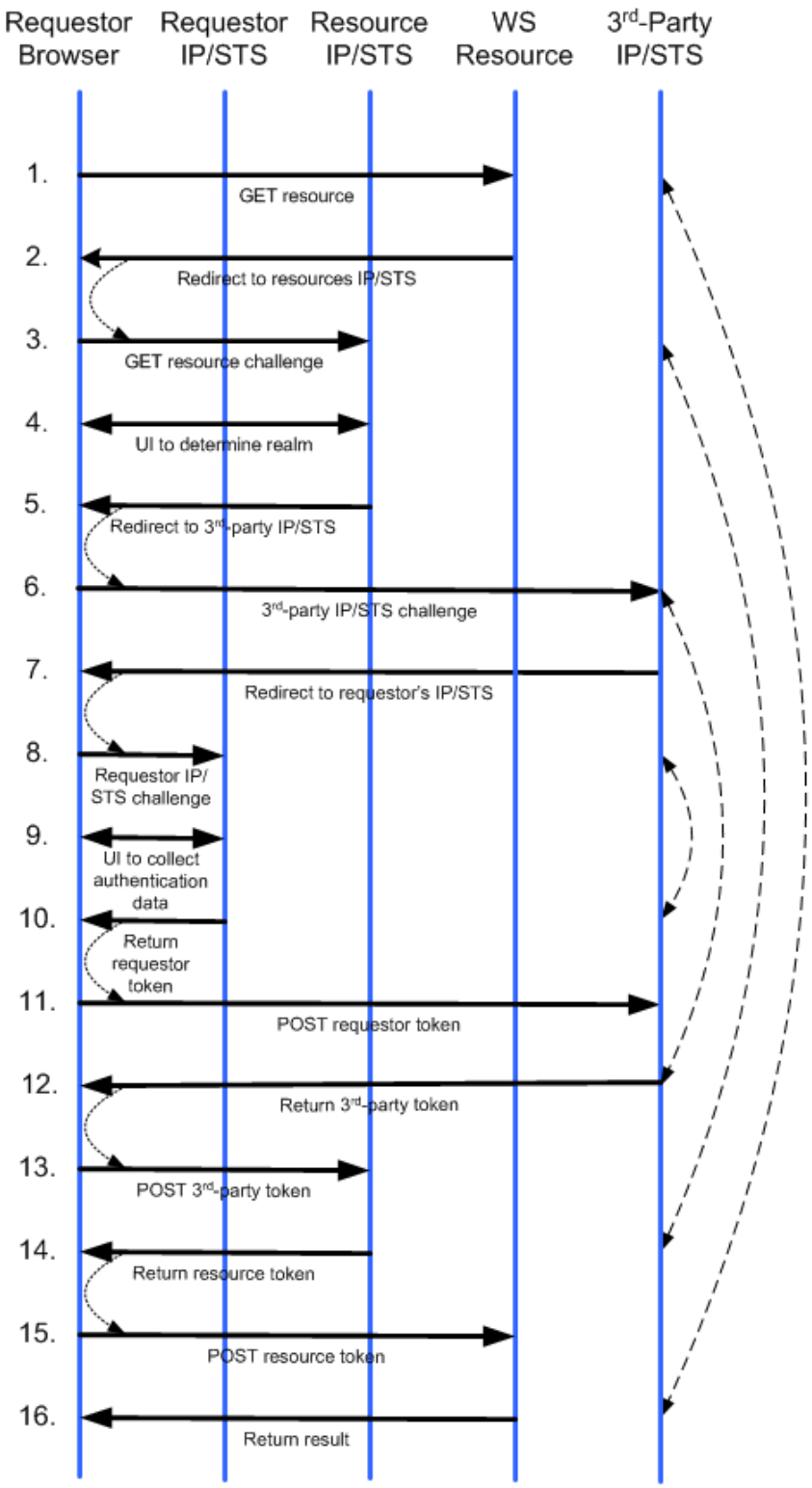
III.9.1. No Resource STS

The figure below illustrates the sign-in scenario above, but without a resource STS. That is, the requestor acts as its own STS:



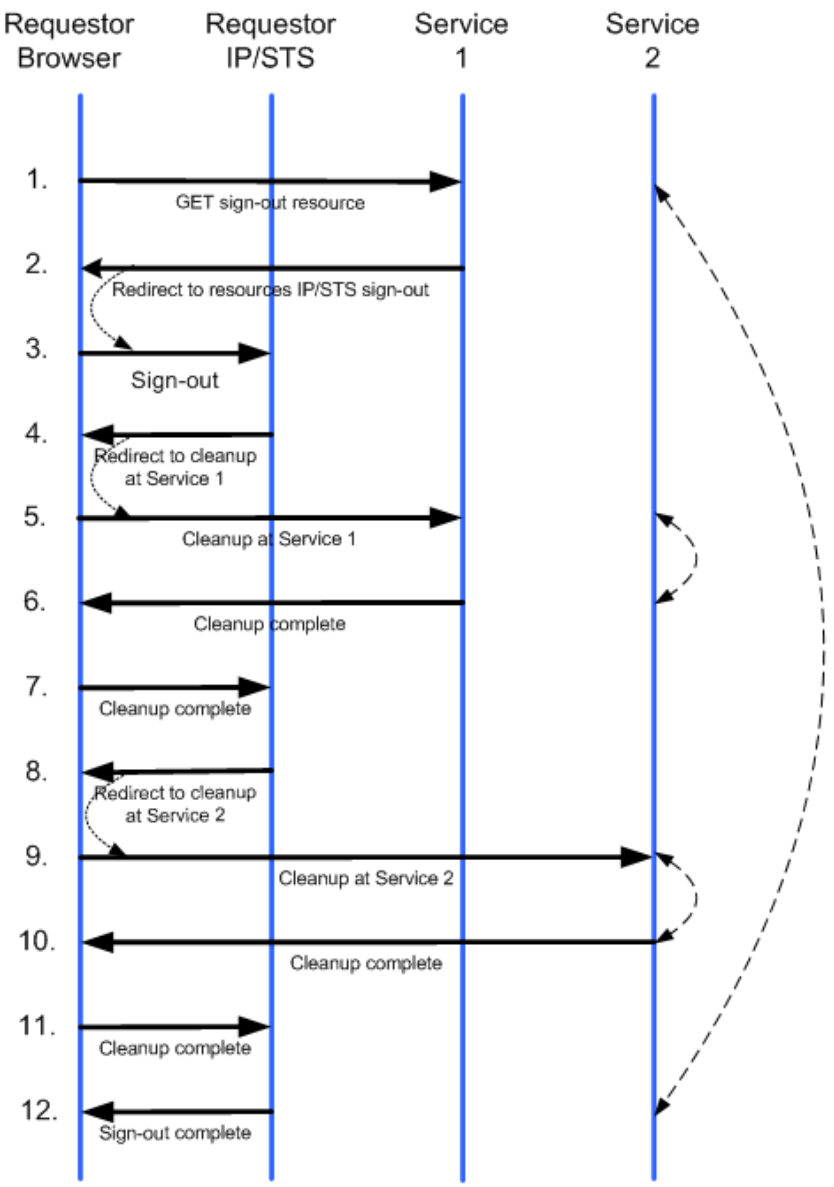
III.9.2. 3rd-Party STS

The figure below illustrates the sign-in scenario above, but trust is brokered through a 3rd-party STS:



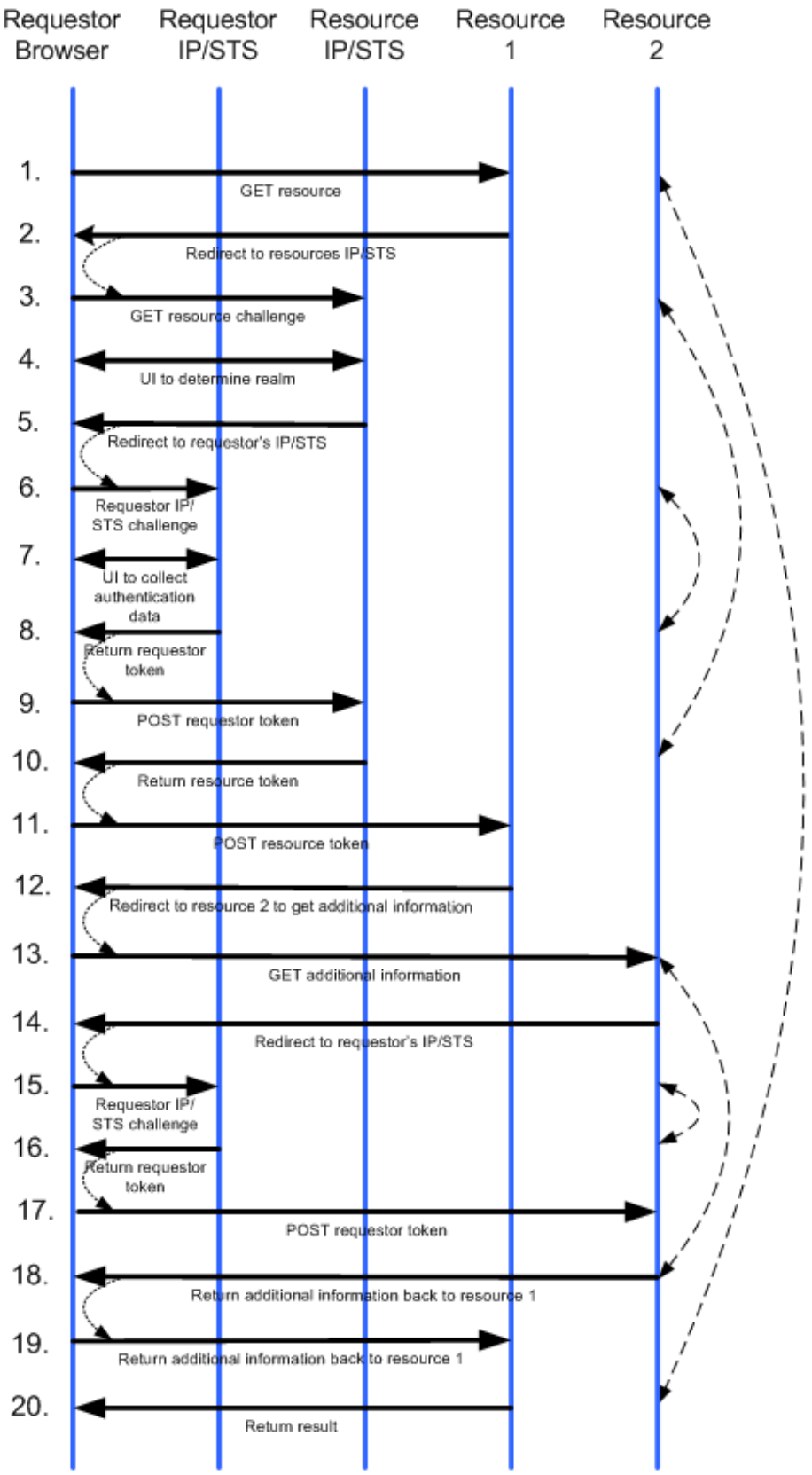
III.9.3. Sign-Out

The figure below illustrates the sign-out flow for a requestor that has signed in at two sites and requests that the sign-out cleanup requests redirect back to the requestor:



III.9.4. Delegated Resource Access

The figure below illustrates the case where a resource accesses data from another resource on behalf of the first resource and the information is returned through the requestor:


Requestor Browser
Requestor IP/STS
Resource IP/STS
Resource 1
Resource 2
1. GET resource
2. Redirect to resources IP/STS
3. GET resource challenge
4. UI to determine realm
5. Redirect to requestor's IP/STS
6. Requestor IP/ STS challenge
7. UI to collect authentication data
8. Return requestor token
9. POST requestor token
10. Return resource token
11. POST resource token
12. Redirect to resource 2 to get additional information
13. GET additional information
14. Redirect to requestor's IP/STS
15. Requestor IP/ STS challenge
16. Return requestor token
17. POST requestor token
18. Return additional information back to resource 1
19. Return additional information back to resource 1
20. Return result