



Web Services Federation Language (WS-Federation) Version 1.2

Committee Draft

June 2008

Specification URIs:

This Version:

<http://docs.oasis-open.org/wsfed/federation/200706/ws-federation-1.2-spec-cd-01.doc>
<http://docs.oasis-open.org/wsfed/federation/200706/ws-federation-1.2-spec-cd-01.pdf>
<http://docs.oasis-open.org/wsfed/federation/200706/ws-federation-1.2-spec-cd-01.html>

Previous Version:

na

Latest Version:

<http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.doc>
<http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.pdf>
<http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html>

Technical Committee:

OASIS Web Services Federation (WSFED) TC

Chair(s):

Chris Kaler, Microsoft
Michael McIntosh, IBM

Editor(s):

Marc Goodner, Microsoft
Anthony Nadalin, IBM

Related work:

This specification is related to:

- WSS
- WS-Trust
- WS-SecurityPolicy

Declared XML Namespace(s):

<http://docs.oasis-open.org/wsfed/federation/200706>
<http://docs.oasis-open.org/wsfed/authorization/200706>
<http://docs.oasis-open.org/wsfed/privacy/200706>

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.

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 document was last revised or approved by the WSFED TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at <http://www.oasis-open.org/committees/wsfed/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/wsfed/ipr.php>).

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/wsfed/>.

Notices

Copyright © OASIS® 2008. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

1	Introduction	8
1.1	Document Roadmap	8
1.2	Goals and Requirements	9
1.2.1	Requirements	9
1.2.2	Non-Goals	10
1.3	Notational Conventions	10
1.4	Namespaces	11
1.5	Schema and WSDL Files	12
1.6	Terminology	12
1.7	Normative References	14
1.8	Non-Normative References	17
1.9	Conformance	17
2	Model	18
2.1	Federation Basics	18
2.2	Metadata Model	21
2.3	Security Model	24
2.4	Trust Topologies and Security Token Issuance	24
2.5	Identity Providers	28
2.6	Attributes and Pseudonyms	28
2.7	Attributes, Pseudonyms, and IP/STS Services	32
3	Federation Metadata	34
3.1	Federation Metadata Document	34
3.1.1	Referencing Other Metadata Documents	35
3.1.2	TokenSigningKeyInfo Element	37
3.1.3	TokenKeyTransferKeyInfo Element	38
3.1.4	IssuerNamesOffered Element	39
3.1.5	TokenIssuerEndpoints Element	39
3.1.6	TokenIssuerMetadata Element	40
3.1.7	PseudonymServiceEndpoints Element	41
3.1.8	AttributeServiceEndpoints Element	41
3.1.9	SingleSignOutSubscriptionEndpoints Element	42
3.1.10	SingleSignOutNotificationEndpoints Element	42
3.1.11	TokenTypesOffered Element	43
3.1.12	ClaimTypesOffered Element	44
3.1.13	ClaimDialectsOffered Element	44
3.1.14	AutomaticPseudonyms Element	45
3.1.15	[Signature] Property	45
3.1.16	PassiveRequestorEndpoints Element	Error! Bookmark not defined.
3.1.17	TargetScopes Element	Error! Bookmark not defined.
3.1.18	ContactInfoAddress Element	Error! Bookmark not defined.
3.1.19	Example Federation Metadata Document	48
3.2	Acquiring the Federation Metadata Document	49

3.2.1	WSDL	50
3.2.2	The Federation Metadata Path	50
3.2.3	Retrieval Mechanisms	50
3.2.4	FederatedMetadataHandler Header	51
3.2.5	Metadata Exchange Dialect	52
3.2.6	Publishing Federation Metadata Location	53
3.2.7	Federation Metadata Acquisition Security	54
4	Sign-Out	55
4.1	Sign-Out Message	55
4.2	Federating Sign-Out Messages	57
5	Attribute Service	59
6	Pseudonym Service	61
6.1	Filtering Pseudonyms	62
6.2	Getting Pseudonyms	63
6.3	Setting Pseudonyms	65
6.4	Deleting Pseudonyms	66
6.5	Creating Pseudonyms	66
7	Security Tokens and Pseudonyms	68
7.1	RST and RSTR Extensions	69
7.2	Username and Passwords	69
7.3	Public Keys	70
7.4	Symmetric Keys	70
8	Additional WS-Trust Extensions	71
8.1	Reference Tokens	71
8.2	Indicating Federations	72
8.3	Obtaining Proof Tokens from Validation	72
8.4	Client-Based Pseudonyms	73
8.5	Indicating Freshness Requirements	74
9	Authorization	75
9.1	Authorization Model	75
9.2	Indicating Authorization Context	75
9.3	Common Claim Dialect	77
9.3.1	Expressing value constraints on claims	79
9.4	Claims Target	81
9.5	Authorization Requirements	82
10	Indicating Specific Policy/Metadata	84
11	Authentication Types	86
12	Privacy	87
12.1	Confidential Tokens	87
12.2	Parameter Confirmation	88
12.3	Privacy Statements	89
13	Web (Passive) Requestors	91
13.1	Approach	91
13.1.1	Sign-On	91

13.1.2 Sign-Out	92
13.1.3 Attributes	93
13.1.4 Pseudonyms	94
13.1.5 Artifacts/Cookies	95
13.1.6 Bearer Tokens and Token References	95
13.1.7 Freshness	95
13.2 HTTP Protocol Syntax	96
13.2.1 Parameters	96
13.2.2 Requesting Security Tokens	97
13.2.3 Returning Security Tokens	99
13.2.4 Sign-Out Request Syntax	100
13.2.5 Attribute Request Syntax	101
13.2.6 Pseudonym Request Syntax	102
13.3 Detailed Example of Web Requester Syntax	102
13.4 Request and Result References	106
13.5 Home Realm Discovery	109
13.5.1 Discovery Service	109
13.6 Minimum Requirements	109
13.6.1 Requesting Security Tokens	109
13.6.2 Returning Security Tokens	110
13.6.3 Details of the RequestSecurityTokenResponse element	110
13.6.4 Details of the Returned Security Token Signature	111
13.6.5 Request and Response References	111
14 Additional Policy Assertions	112
14.1 RequireReferenceToken Assertion	112
14.2 WebBinding Assertion	113
14.3 Authorization Policy	114
15 Error Handling	115
16 Security Considerations	117
Appendix A WSDL	119
Appendix B Sample HTTP Flows for Web Requestor Detailed Example	120
Appendix C Sample Use Cases	123
C.1 Single Sign On	123
C.2 Sign-Out	124
C.3 Attributes	124
C.4 Pseudonyms	125
C.5 Detailed Example	126
C.6 No Resource STS	127
C.7 3 rd -Party STS	128
C.8 Delegated Resource Access	128
C.9 Additional Web Examples	129
No Resource STS	129
3 rd -Party STS	130
Sign-Out	131

Delegated Resource Access	132
Appendix D SAML Binding of Common Claims	134
Appendix E Acknowledgements	135
Appendix F Revision History	Error! Bookmark not defined.

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. The assertions defined within this specification have been designed to work independently of a specific version of WS-Policy. At the time of the publication of this specification the versions of WS-Policy known to correctly compose with this specification are WS-Policy 1.2 and 1.5. Within this specification the use of the namespace prefix `wsp` refers generically to the WS-Policy namespace, not a specific version.

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.

48 Chapter 7 presents how pseudonyms can be directly integrated into security token services by extending
49 the token request and response messages defined in WS-Trust.
50 Chapter 8 introduces additional extensions to WS-Trust that are designed to facilitate federation and
51 includes the use of token references, federation selection, extraction of keys for different trust styles, and
52 different authentication types.
53 Chapter 9 describes federated authorization including extensions to WS-Trust and minimum
54 requirements.
55 Chapter 10 describes how specific policy and metadata can be provided for a specific message pattern
56 and during normal requestor/recipient interactions.
57 Chapter 11 describes pre-defined types of authentication for use with WS-Trust.
58 Chapter 12 describes extensions to WS-Trust for privacy of security token claims and how privacy
59 statements can be made in federated metadata documents.
60 Chapter 13 describes how WS-Federation and WS-Trust can be used by web browser requestors and
61 web applications that do not support direct SOAP messaging.
62 Chapter 14 describes extensions to WS-SecurityPolicy to allow federation participants to indicate
63 additional federation requirements.
64 Chapters 15 and 16 define federation-specific error codes and outline security considerations for
65 architects, implementers, and administrators of federated systems.
66 Chapters 17 and 18 acknowledge contributors to the specification and all references made by this
67 specification to other documents.
68 Appendix I provides a sample WSDL definition of the services defined in this specifications.
69 Appendix II provides a detailed example of the messages for a Web browser-based requestor that is
70 using the federation mechanisms described in chapter 9.
71 Appendix III describes several additional use cases motivating the federation framework for both SOAP-
72 based and Web browser-based requestors.

73 **1.2 Goals and Requirements**

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

76 **1.2.1 Requirements**

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

- 78 • Enable appropriate sharing of identity, authentication, and authorization data using different or like
79 mechanisms
- 80 • Allow federation using different types of security tokens, trust topologies, and security infrastructures
- 81 • Facilitate brokering of trust and security token exchange for both SOAP requestors and Web
82 browsers using common underlying mechanisms and semantics
- 83 • Express federation metadata to facilitate communication and interoperability between federation
84 participants
- 85 • Allow identity mapping to occur at either requestor, target service, or any IP/STS
- 86 • Provide identity mapping support if target services choose to maintain OPTIONAL local identities, but
87 do not require local identities
- 88 • Allow for different levels of privacy for identity (e.g. different forms and uniqueness of digital identities)
89 information and attributes
- 90 • Allow for authenticated but anonymous federation

91 1.2.2 Non-Goals

92 The following topics are outside the scope of this document:

- 93 • Definition of message security (see WS-Security)
- 94 • Trust establishment/verification protocols (see WS-Trust)
- 95 • Management of trust or trust relationships
- 96 • Specification of new security token formats beyond token references
- 97 • Specification of new attribute store interfaces beyond UDDI
- 98 • Definition of new security token assertion/claim formats
- 99 • Requirement on specific security token formats
- 100 • Requirement on specific types of trust relationships
- 101 • Requirement on specific types of account linkages
- 102 • Requirement on specific types of identity mapping

103 1.3 Notational Conventions

104 The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD
105 NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described
106 in [KEYWORDS].

107 This specification uses the following syntax to define outlines for assertions:

- 108 • The syntax appears as an XML instance, but values in italics indicate data types instead of literal
109 values.
- 110 • Characters are appended to elements and attributes to indicate cardinality:
 - 111 ○ "?" (0 or 1)
 - 112 ○ "*" (0 or more)
 - 113 ○ "+" (1 or more)
- 114 • The character "|" is used to indicate a choice between alternatives.
- 115 • The characters "(" and ")" are used to indicate that contained items are to be treated as a group
116 with respect to cardinality or choice.
- 117 • The characters "[" and "]" are used to call out references and property names.
- 118 • Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be
119 added at the indicated extension points but MUST NOT contradict the semantics of the parent
120 and/or owner, respectively. By default, if a receiver does not recognize an extension, the receiver
121 SHOULD ignore the extension; exceptions to this processing rule, if any, are clearly indicated
122 below.
- 123 • XML namespace prefixes (see Table 2) are used to indicate the namespace of the element being
124 defined.

125

126 Elements and Attributes defined by this specification are referred to in the text of this document using
127 XPath 1.0 expressions. Extensibility points are referred to using an extended version of this syntax:

- 128 • An element extensibility point is referred to using {any} in place of the element name. This
129 indicates that any element name can be used, from any namespace other than the namespace of
130 this specification.
- 131 • An attribute extensibility point is referred to using @{any} in place of the attribute name. This
132 indicates that any attribute name can be used, from any namespace other than the namespace of
133 this specification.

134 Extensibility points in the exemplar may not be described in the corresponding text.

135 1.4 Namespaces

136 The following namespaces are used in this document:

Prefix	Namespace
fed	http://docs.oasis-open.org/wsfed/federation/200706
auth	http://docs.oasis-open.org/wsfed/authorization/200706
priv	http://docs.oasis-open.org/wsfed/privacy/200706
mex	http://schemas.xmlsoap.org/ws/2004/09/mex
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
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-wssecurity-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

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

139 1.5 Schema and WSDL Files

140 The schemas for this specification can be located at:

```
141 http://docs.oasis-open.org/wsfed/federation/200706/federation.xsd  
142 http://docs.oasis-open.org/wsfed/authorization/200706/authorization.xsd  
143 http://docs.oasis-open.org/wsfed/privacy/200706/privacy.xsd
```

144 The WSDL for this specification can be located at:

```
145 http://docs.oasis-open.org/wsfed/federation/200706/federation.wsdl
```

146 1.6 Terminology

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

188 **Identity Provider (IP)** – An *Identity Provider* is an entity that acts as an authentication service to end
189 requestors and a data origin authentication service to service providers (this is typically an extension of a
190 Security Token Service). Identity Providers (IP) are trusted (logical) 3rd parties which need to be trusted
191 both by the requestor (to maintain the requestor's identity information as the loss of this information can
192 result in the compromise of the requestors identity) and the service provider which MAY grant access to
193 valuable resources and information based upon the integrity of the identity information provided by the IP.

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

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

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

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

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

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

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

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

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

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

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

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

223 **Security Token Service (STS)** - A *Security Token Service* is a Web service that provides issuance and
224 management of security tokens (see [WS-Security] for a description of security tokens). That is, it
225 makes security statements or claims often, although not required to be, in cryptographically protected
226 sets. These statements are based on the receipt of evidence that it can directly verify, or security tokens
227 from authorities that it trusts. To assert trust, a service might prove its right to assert a set of claims by
228 providing a security token or set of security tokens issued by an STS, or it could issue a security token
229 with its own trust statement (note that for some security token formats this can just be a re-issuance or
230 co-signature). This forms the basis of trust brokering.

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

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

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

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

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

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

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

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

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

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

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

265 1.7 Normative References

266 [HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T.
 267 Berners-Lee, RFC 2616, "Hypertext Transfer Protocol -- HTTP/1.1".
 268 June 1999.
 269 <http://ietf.org/rfc/rfc2616.txt>

270 [HTTPS] IETF Standard, "The TLS Protocol", January 1999.
 271 <http://www.ietf.org/rfc/rfc2246.txt>

272 [KEYWORDS] S. Bradner, "Key words for use in RFCs to Indicate Requirement
 273 Levels", RFC 2119, Harvard University, March 1997.
 274 <http://www.ietf.org/rfc/rfc2119.txt>.

275 [SOAP] W3C Note, "SOAP: Simple Object Access Protocol 1.1", 08 May 2000.
 276 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

277	[SOAP12]	W3C Recommendation, "SOAP 1.2 Part 1: Messaging Framework (Second Edition)", 27 April 2007.
278		
279		http://www.w3.org/TR/2007/REC-soap12-part1-20070427/
280	[URI]	T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 3986, MIT/LCS, Day Software, Adobe Systems, January 2005.
281		
282		
283		http://www.ietf.org/rfc/rfc3986.txt
284	[WS-Addressing]	W3C Recommendation, "Web Services Addressing (WS-Addressing)", 9 May 2006.
285		
286		http://www.w3.org/TR/2006/REC-ws-addr-core-20060509
287	[WS-Eventing]	W3C Member Submission, "Web Services Eventing (WS-Eventing)", 15 March 2006
288		
289		http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/
290	[WS-MetadataExchange]	Web Services Metadata Exchange (WS-MetadataExchange), August 2006
291		
292		http://schemas.xmlsoap.org/ws/2004/09/mex/
293	[WS-Policy]	W3C Member Submission "Web Services Policy 1.2 - Framework", 25 April 2006.
294		
295		http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/
296		W3C Recommendation "Web Services Policy 1.5 – Framework", 04 September 2007
297		
298		http://www.w3.org/TR/2007/REC-ws-policy-20070904/
299	[WS-PolicyAttachment]	W3C Member Submission "Web Services Policy 1.2 - Attachment", 25 April 2006.
300		
301		http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/
302		
303		W3C Recommendation "Web Services Policy 1.5 – Attachment", 04 September 2007
304		
305		http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904/
306	[WS-SecurityPolicy]	OASIS Standard, "WS-SecurityPolicy 1.2", July 2007
307		http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702
308	[WS-Security]	OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", March 2004.
309		
310		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
311		
312		OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", February 2006.
313		
314		http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
315		
316	[WSS:UsernameToken]	OASIS Standard, "Web Services Security: UsernameToken Profile", March 2004
317		
318		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
319		
320		OASIS Standard, "Web Services Security: UsernameToken Profile 1.1", February 2006
321		

322		http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf
323		
324	[WSS:X509Token]	OASIS Standard, "Web Services Security X.509 Certificate Token Profile", March 2004
325		
326		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
327		
328		OASIS Standard, "Web Services Security X.509 Certificate Token Profile", February 2006
329		
330		http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf
331		
332	[WSS:KerberosToken]	OASIS Standard, "Web Services Security Kerberos Token Profile 1.1", February 2006
333		
334		http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf
335		
336	[WSS:SAMLTokenProfile]	OASIS Standard, "Web Services Security: SAML Token Profile", December 2004
337		
338		http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf
339		OASIS Standard, "Web Services Security: SAML Token Profile 1.1", February 2006
340		
341		http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf
342		
343	[WS-ResourceTransfer]	"Web Services Resource Transfer (WS-ResourceTransfer)", August 2006
344		
345		http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/
346	[WS-Transfer]	W3C Member Submission, "Web Services Transfer (WS-Transfer)", 27 September 2006
347		
348		http://www.w3.org/Submission/2006/SUBM-WS-Transfer-20060927/
349	[WS-Trust]	OASIS Standard, "WS-Trust 1.3", March 2007
350		http://docs.oasis-open.org/ws-sx/ws-trust/200512
351	[ISO8601]	ISO Standard 8601:2004(E), "Data elements and interchange formats – Information interchange - Representation of dates and times", Third edition, December 2004
352		
353		
354		http://isotc.iso.org/livelink/livelink/4021199/ISO_8601_2004_E.zip?func=doc.Fetch&nodeid=4021199
355		
356	[DNS-SRV-RR]	Gulbrandsen, et al, RFC 2782, "DNS SRV RR", February 2000.
357		http://ietf.org/rfc/rfc2782.txt
358	[XML-Schema1]	W3C Recommendation, "XML Schema Part 1: Structures Second Edition", 28 October 2004.
359		
360		http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/
361	[XML-Schema2]	W3C Recommendation, "XML Schema Part 2: Datatypes Second Edition", 28 October 2004.
362		
363		http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/
364	[XML-C14N]	W3C Recommendation, "Canonical XML Version 1.0", 15 March 2001
365		http://www.w3.org/TR/2001/REC-xml-c14n-20010315

366	[XML-Signature]	W3C Recommendation, "XML-Signature Syntax and Processing", 12
367		February 2002
368		http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/
369	[WSDL 1.1]	W3C Note, "Web Services Description Language (WSDL 1.1)," 15
370		March 2001.
371		http://www.w3.org/TR/2001/NOTE-wsdl-2001031
372	[XPATH]	W3C Recommendation "XML Path Language (XPath) Version 1.0", 16
373		November 1999.
374		http://www.w3.org/TR/1999/REC-xpath-19991116
375	[RFC 4648]	S. Josefsson, et. al, RFC 4648 "The Base16, Base32, and Base64
376		Data Encodings" October 2006
377		http://www.ietf.org/rfc/rfc4648.txt

378 1.8 Non-Normative References

379

380 1.9 Conformance

381 An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level
382 requirements defined within this specification. A SOAP Node MUST NOT use the XML namespace
383 identifier for this specification (listed in Section 1.4) within SOAP Envelopes unless it is compliant with this
384 specification.

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

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

395 If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for
396 any other unrecognized/unsupported message. If an OPTIONAL message is supported, then the
397 implementation MUST satisfy all of the MUST and REQUIRED sections of the message.

398 2 Model

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

401 2.1 Federation Basics

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

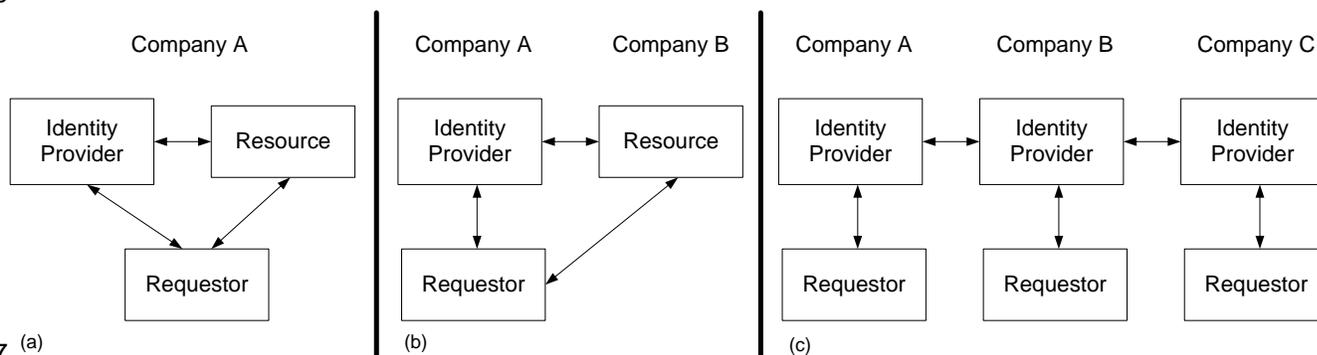
405 In the context of web services the goal is to allow these identities and attributes to be brokered from
406 identity and security token issuers to services and other relying parties without requiring user intervention
407 (unless specified by the underlying policies). This process involves the sharing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

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

413 Federations can exist within organizations and companies as well as across organizations and
414 companies. They can also be ad-hoc collections of principals that choose to participate in a community.

415 The figure below illustrates a few sample federations:

416



417 (a)

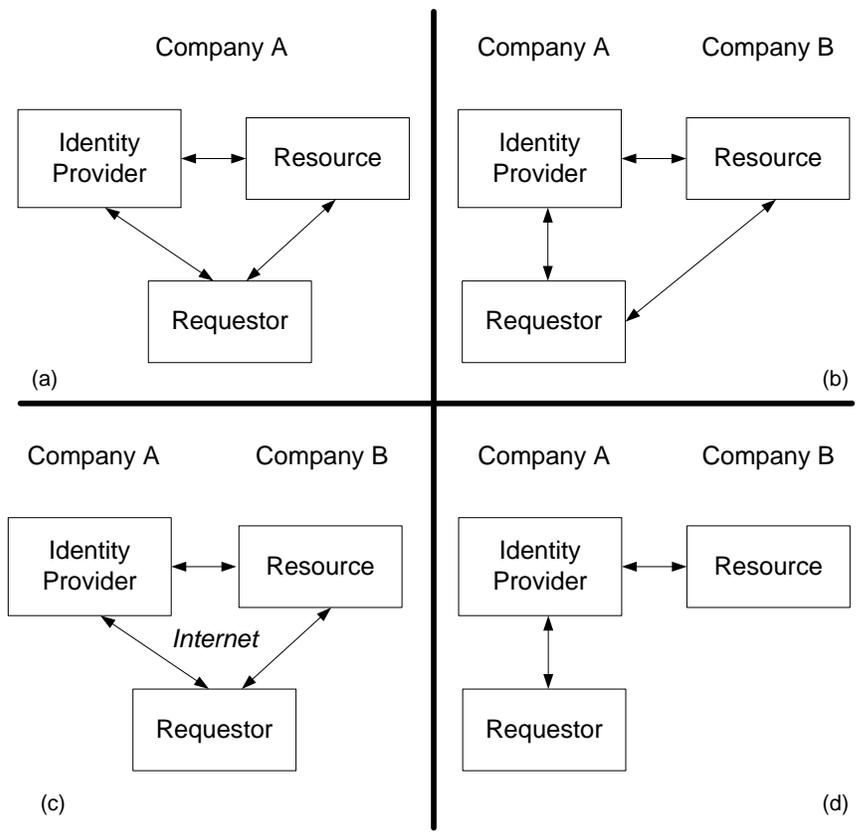
(b)

(c)

418

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

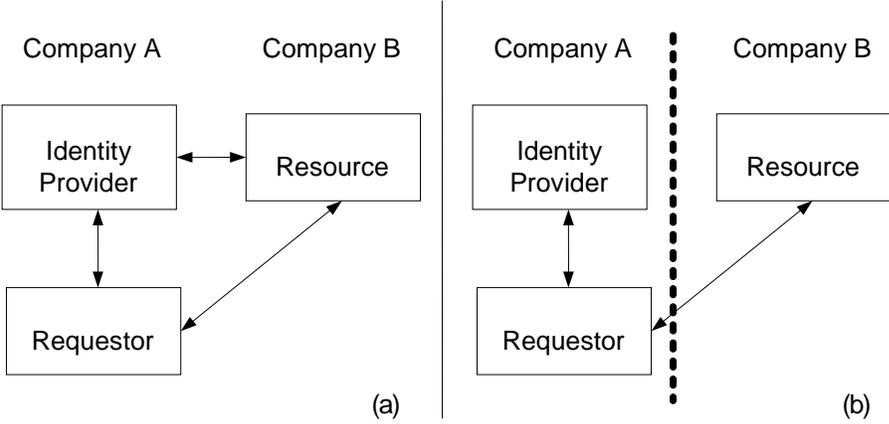
419 As a consequence, federations MAY exist within one or multiple administrative domains, span multiple
420 security domains, and MAY be explicit (requestor knows federation is occurring) or implicit (federation is
421 hidden such as in a portal) as illustrated in the figure below:



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

422
 423
 424
 425
 426
 427
 428

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:

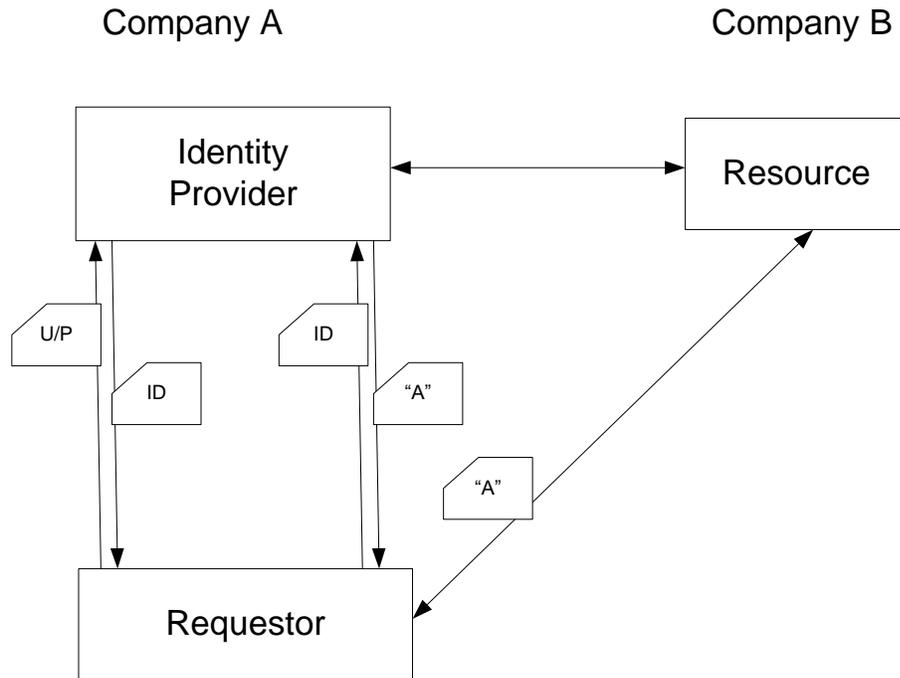


Figures 3a, 3b: Accessibility of Identity Provider

429
 430
 431
 432
 433

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,

434 addresses, etc. In some cases the only information that is exchanged is an authentication statement (e.g.
435 employee of company “A”) allowing the actual requestor to be anonymous as in the example below:



436

437

Figure 4: Sample Anonymous Access

438 To establish a federation context for a principal either the principal's identity is universally accepted (so
439 that its association is “pre-established” across trust realms within a federation context), or it must be
440 brokered into a trusted identity relevant to each trust realm within the federation context. The latter case
441 requires the process of identity mapping – that is, the conversion of a digital identity from one realm to a
442 digital identity valid in another realm by a party that trusts the starting realm and has the rights to speak
443 for (make assertions to) the ending realm, or make assertions that the ending realm trusts. Identity
444 mapping (this brokering) is typically implemented by an IP/STS when initially obtaining tokens for a
445 service or when exchanging tokens at a service's IP/STS.

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

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

457 A third approach is to require the service to be responsible for the identity mapping. That is, the service is
458 given an opaque handle which it must then have mapped into an identity it understands – assuming it
459 cannot directly process the opaque handle. More specifically, the requestor's IP/STS generates a digital
460 identity that cannot be reliably used by the target service as a key for local identity mapping (e.g. the
461 marker is known to be random or the marker's randomness is not known. The target service then uses

462 the requestor's mapping service (called a pseudonym service) to map the given (potentially random)
463 digital identity into a constant service-specific digital identity which it has registered with the requestor's
464 mapping service. This also addresses the collusion issue but pushes the mapping burden onto the
465 service (but keeps the privacy of all information in the requestor's control).

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

468 2.2 Metadata Model

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

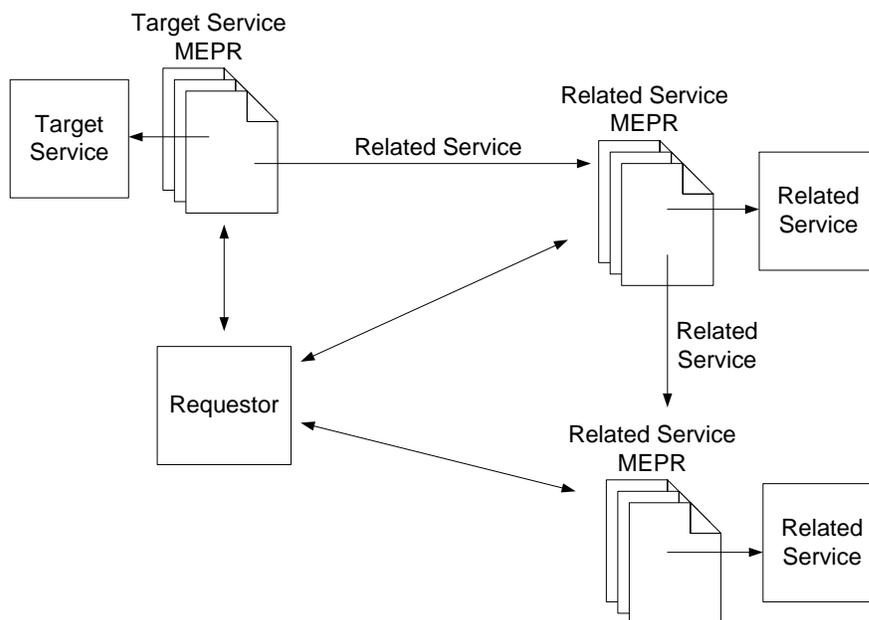
472 This discovery process begins with the target service, that is, the service to which the requester wishes to
473 ultimately communicate. Given the metadata endpoint reference (MEPR) for the target service allows the
474 requestor to obtain all requirement metadata about the service (e.g. federation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

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

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

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

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



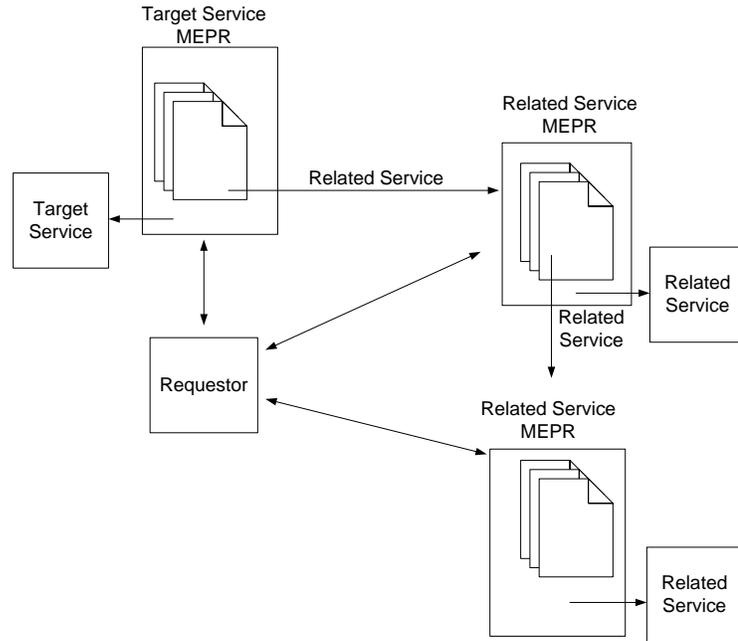
492

493

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

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

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



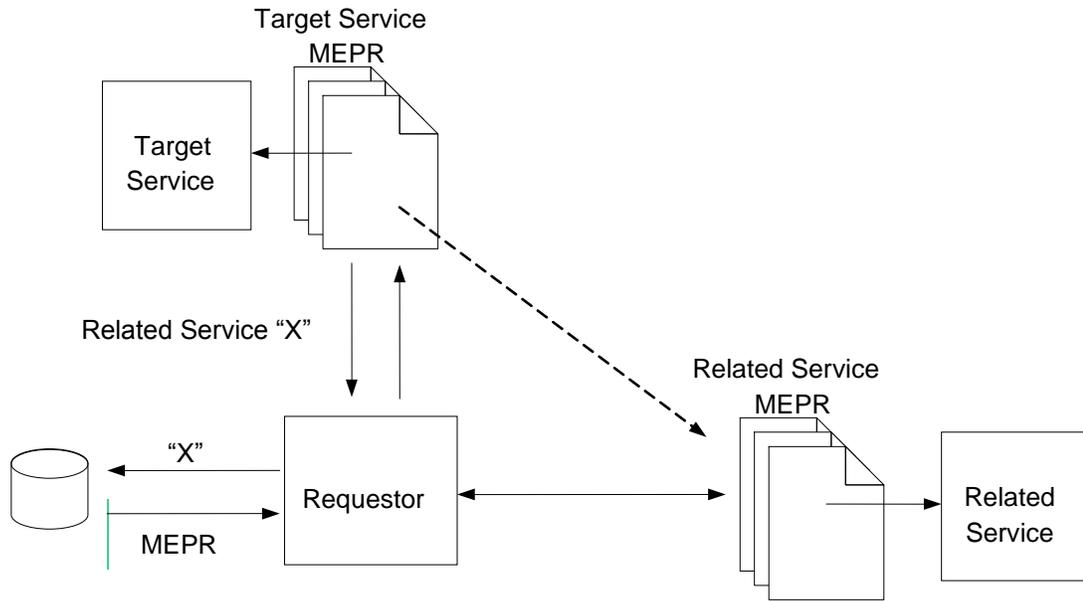
500

501

Figure 5b: Obtaining Federation Metadata (embedded)

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

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



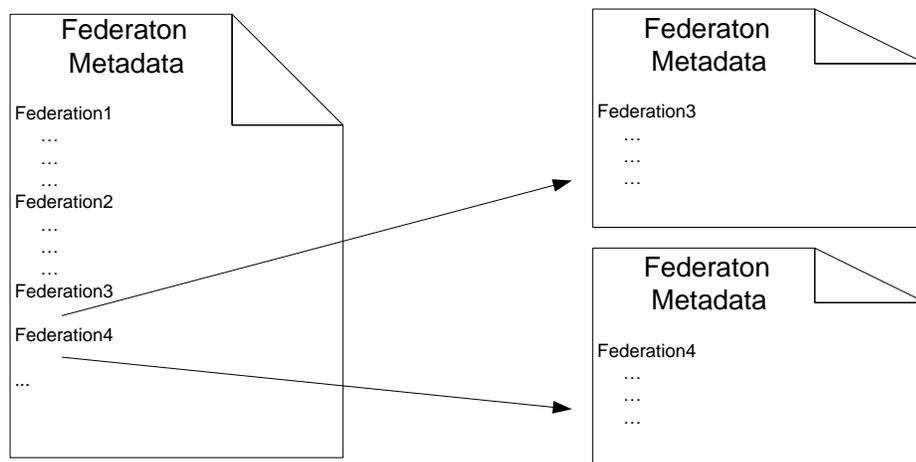
511

512

Figure 6: Example of Logical Service Names

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

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



523

524

Figure 7: Federation Metadata Document

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

528 **2.3 Security Model**

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

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

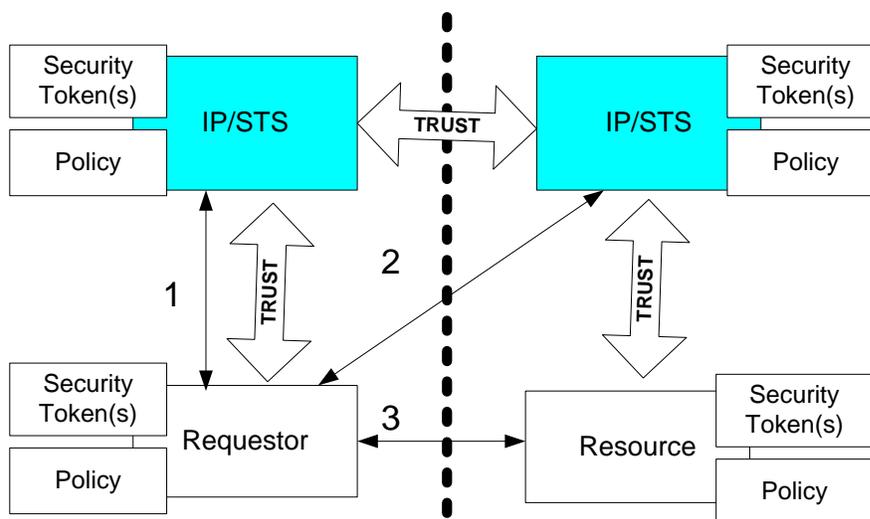
540 These initial security tokens MAY be accepted by various Web services or exchanged at Security Token
541 Services (STS) / Identity Providers (IP) for additional security tokens subject to established trust
542 relationships and trust policies as described in WS-Trust. This exchange can be used to create a local
543 access token or to map to a local identity.

544 This specification also describes an Attribute/Pseudonym service that can be used to provide
545 mechanisms for restricted sharing of principal information and principal identity mapping (when different
546 identities are used at different resources). The metadata mechanisms described in this document are
547 used to enable a requestor to discover the location of various Attribute/Pseudonym services.

548 Finally, it should be noted that just as a resource MAY act as its own IP/STS or have an embedded
549 IP/STS. Similarly, a requestor MAY also act as its own IP/STS or have an embedded IP/STS.

550 **2.4 Trust Topologies and Security Token Issuance**

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



557

558

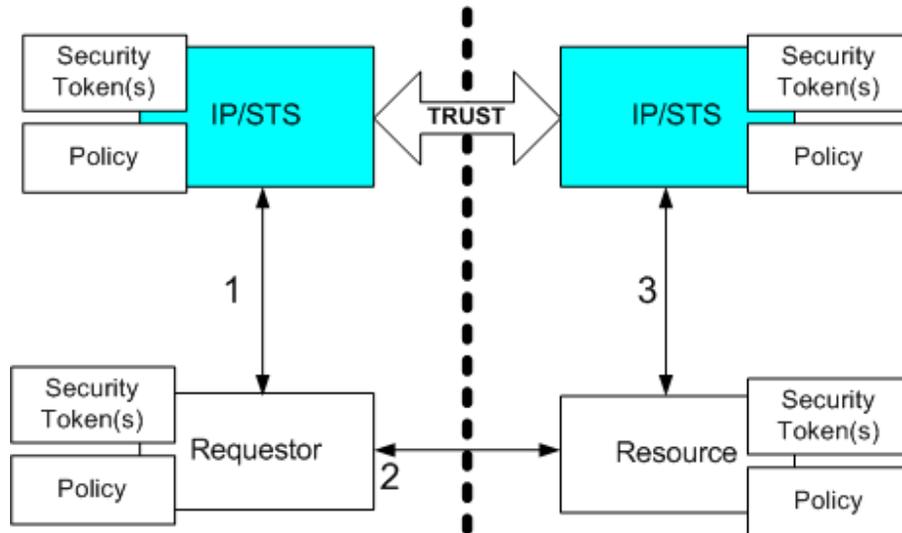
Figure 8: Federation and Trust Model

559 Figure 8 above illustrates one way the WS-Trust model may be applied to simple federation scenarios.
560 Here security tokens (1) from the requestor's trust realm are used to acquire security tokens from the

561 resource's trust realm (2) These tokens are then presented to the resource/service's realm (3) to access
562 the resource/service . That is, a token from one STS is exchanged for another at a second STS or
563 possibly stamped or cross-certified by a second STS (note that this process can be repeated allowing for
564 trust chains of different lengths).

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

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



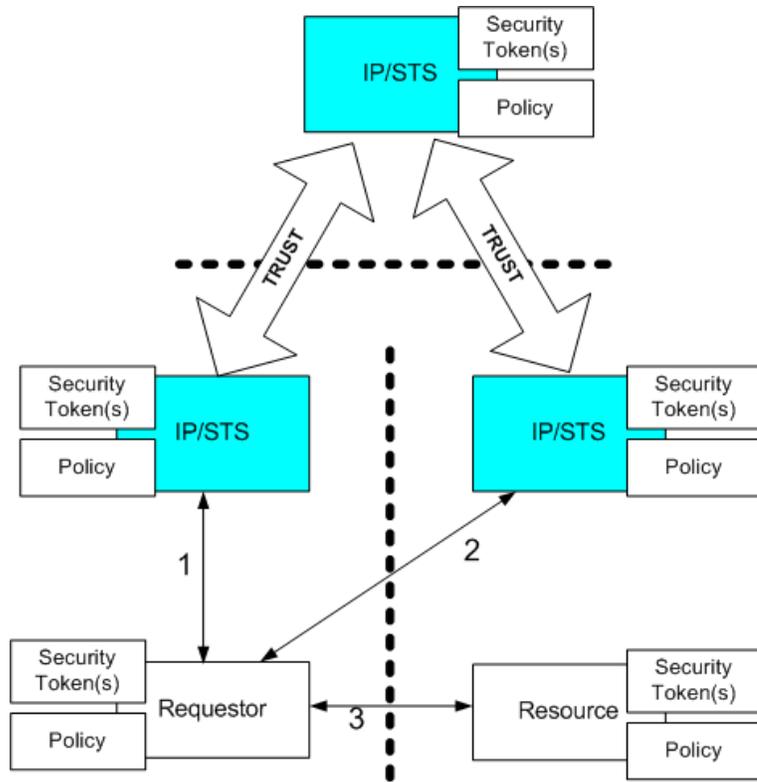
572

573

Figure 9: Alternate Federation and Trust Model

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

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



580

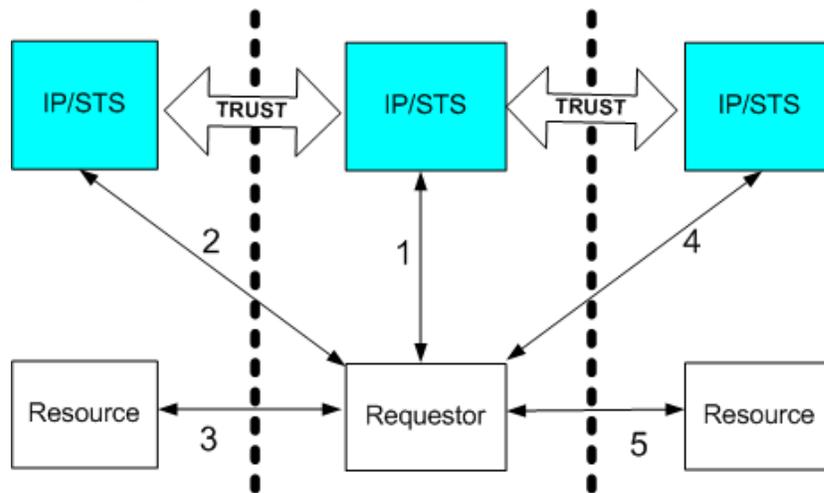
581

Figure 10: Indirect Trust

582

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:

583



584

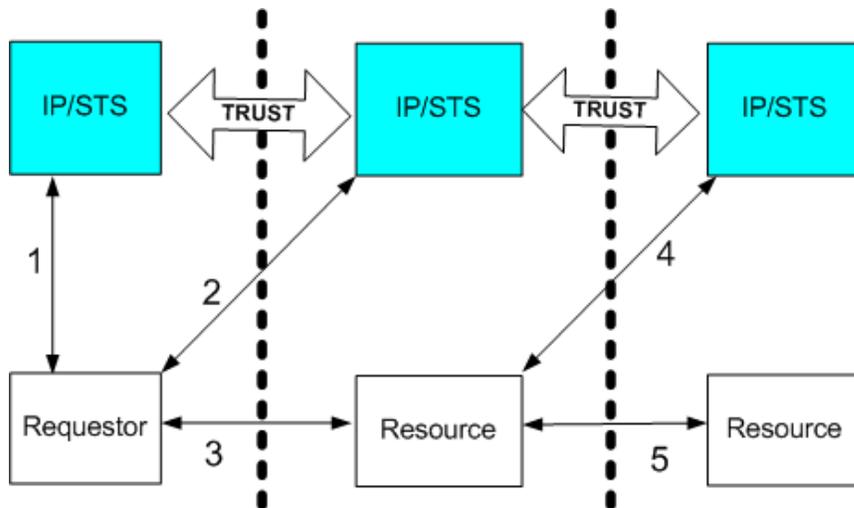
585

Figure 11: Multiple Trust Domains

586

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:

587

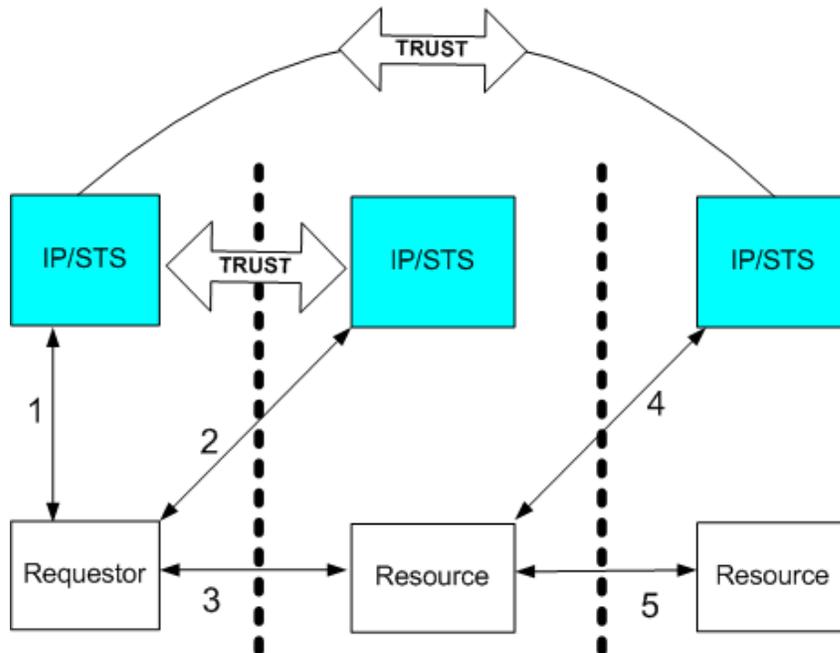


588

589

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

590 In such cases (as illustrated in Figure 12) the first resource, in its capacity as a second requestor on
 591 behalf of the original requestor, provides security tokens to allow/indicate proof of (ability for) delegation.
 592 It should be noted that there are a number of variations on this scenario. For example, the security token
 593 service for the final resource may only have a trust relationship with the token service from the original
 594 requestor (illustrated below), as opposed to the figure above where the trust doesn't exist with the original
 595 requestor's STS.



596

597

Figure 13: No Trust Relationship between Resource Providers

598 Specifically, in Figure 13 the resource or resource's security token service initiates a request for a security
 599 token that delegates the required claims. For more details on how to format such requests, refer to WS-
 600 Trust. These options are specified as part of the <wst:RequestSecurityToken> request.

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

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

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

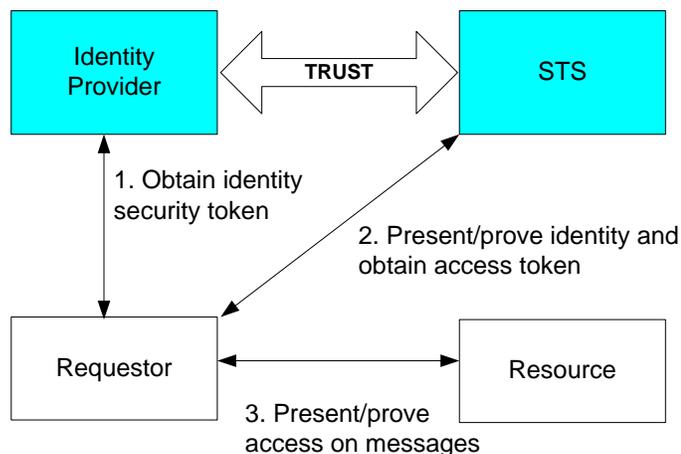
609 2.5 Identity Providers

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

615 One possible function of a security token service is to provide digital identities – an *Identity Provider (IP)*.
616 This is a special type of security token service that, at a minimum, performs authentication and can make
617 identity (or origin) claims in issued security tokens.

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

620 The following example illustrates a possible combination of an Identity Provider (IP) and STS. In Figure
621 14, a requestor obtains an identity security token from its Identity Provider (1) and then presents/proves
622 this to the STS for the desired resource. If successful (2), and if trust exists and authorization is
623 approved, the STS returns an access token to the requestor. The requestor then uses the access token
624 on requests to the resource or Web service (3). Note that it is assumed that there is a trust relationship
625 between the STS and the identity provider.



626

627

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

628 2.6 Attributes and Pseudonyms

629 Attributes are typically used when applications need additional information about the requestor that has
630 not already been provided or cached, or is not appropriate to be sent in every request or saved in security
631 tokens. Attributes are also used when ad hoc information is needed that cannot be known at the time the
632 requests or token issuance.

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

637 When requestors interact with resources in different trust realms (or different parts of a federation), there
 638 is often a need to *know* additional information about the requestor in order to authorize, process, or
 639 personalize the experience. A service, known as an *Attribute Service* MAY be available within a realm or
 640 federation. As such, an attribute service is used to provide the attributes about a requestor that are
 641 relevant to the completion of a request, given that the service is authorized to obtain this information.
 642 This approach allows the sharing of data between authorized entities.

643 To facilitate single sign-on where multiple identities need to be automatically mapped and the privacy of
 644 the principal needs to be maintained, there MAY also be a *pseudonym service*. A pseudonym service
 645 allows a principal to have different *aliases* at different resources/services or in different realms, and to
 646 optionally have the pseudonym change per-service or per-login. While some scenarios support identities
 647 that are trusted as presented, pseudonyms services allow those cases where identity mapping needs to
 648 occur between an identity and a pseudonym on behalf of the principal.

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

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

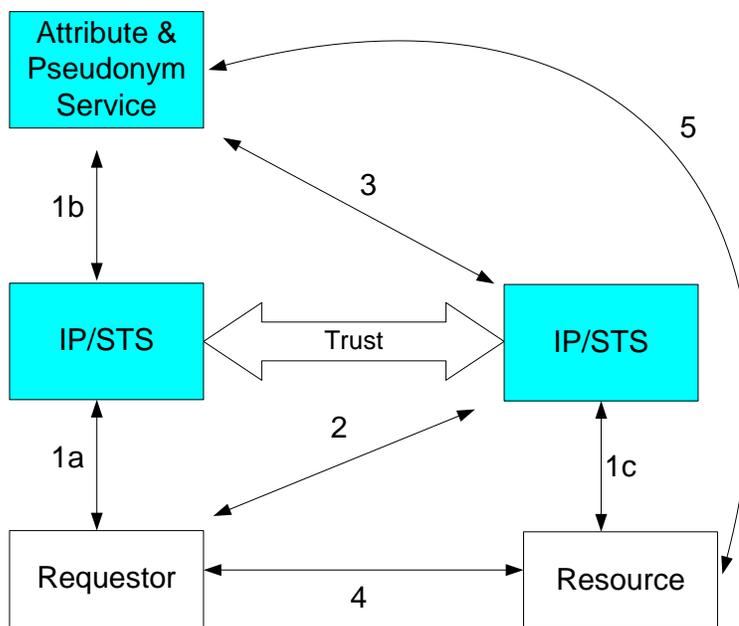


Figure 15: Attributes & Pseudonyms

660 With respect to Figure 15, in an initial (bootstrap) case, a requestor has knowledge of the policies of a
 661 resource, including its IP/STS. The requestor obtains its identity token from its IP/STS (1a) and
 662 communicates with the resource's IP/STS (2) to obtain an access token for the resource. In this example
 663 the resource IP/STS has registered a pseudonym with the requestor's pseudonym service (3) possibly for
 664 sign-out notification or for service-driven mappings. The requestor accesses the resource using the
 665 pseudonym token (4). The resource can obtain additional information (5) from the requestor's attribute
 666 service if authorized based on its identity token (1c). It should be noted that trust relationships will need
 667 to exist in order for the resource or its IP/STS to access the requestor's attribute or pseudonym service.
 668 In subsequent interactions, the requestor's IP/STS may automatically obtain pseudonym credentials for
 669 the resource (1b) if they are available. In such cases, steps 2 and 3 are omitted. Another possible

670 scenario is that the requestor registers the tokens from step 2 with its pseudonym service directly (not
671 illustrated). Note that if the mapping occurs at the IP/STS then a service-consumable identity is returned
672 in step 1a.

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

675 Pseudonyms are an OPTIONAL mechanism that can be used by authorized cooperating services to
676 federate identities and securely and safely access profile attribute information, while protecting the
677 principal's privacy. This is done by allowing services to issue pseudonyms for authenticated identities
678 and letting authorized services query for profile attributes which they are allowed to access, including
679 pseudonyms specific to the requesting service. The need for service-driven mapping is typically known
680 up-front or indicated in metadata.

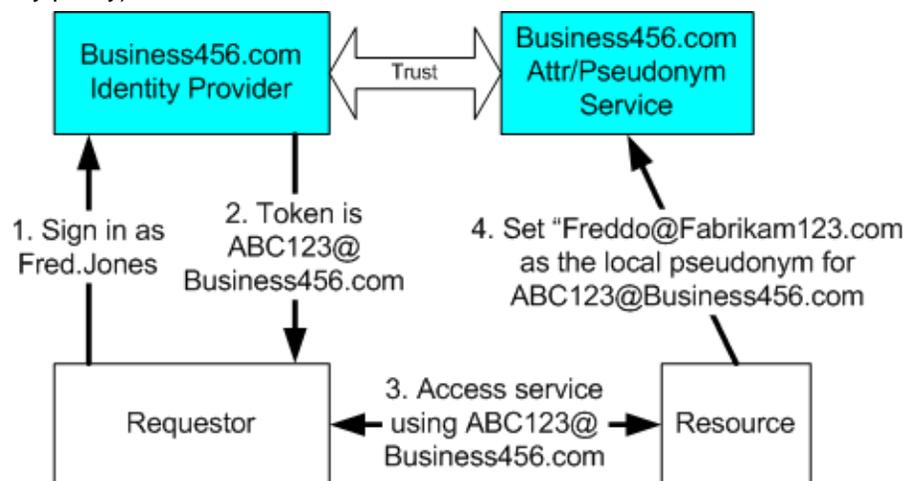
681 While pseudonyms are helpful for principals who want to keep from having their activities tracked
682 between the various sites they visit, they may add a level of complexity as the principal must typically
683 manage the authorization and privacy of each pseudonym. For principals who find this difficult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

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

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

697 In this example the unique identity returned is "ABC123@Business456.com". The requestor then visits
698 Fabrikam123.com. The Web service at Fabrikam123.com can request information about the requestor
699 "ABC123@Business456.com" from the pseudonym/attribute service for Business456.com. If the
700 requester has authorized it, the information will be provided by the identity service.

701 A variation on this first approach is the use of randomly generated pseudonyms; the requestor may
702 indicate that they are "Freddo" to the Web service at Fabrikam123.com through some sort of mapping.
703 Fabrikam123.com can now inform the pseudonym service for Business456.com that
704 "ABC123@Business456.com" is known as "Freddo@Fabrikam123.com" (if authorized and allowed by the
705 principal's privacy policy). This is illustrated below:



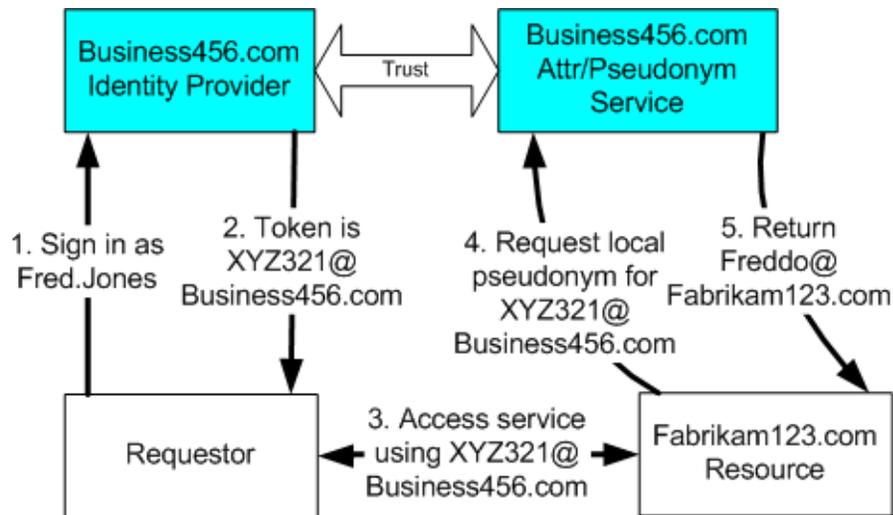
706

707

Figure 16: Pseudonym

708 Note that the attribute, pseudonym, and Identity Provider services could be combined or separated in
709 many different configurations. Figure 16 illustrates a configuration where the IP is separate from the
710 pseudonym service. In such a case there is shared information or specialized trust to allow the
711 pseudonym service to perform the mapping or to make calls to the IP to facilitate the mapping. Different
712 environments will have different configurations based on their needs, security policies, technologies used,
713 and existing infrastructure.

714 The next time the requestor signs in to Business456.com Identity Provider, it might return a new identifier,
715 like XYZ321@Business456.com, in the token to be presented to Fabrikam in step 3. The Web service at
716 Fabrikam123.com can now request a local pseudonym for XYZ321@Business456.com and be told
717 "Freddo@Fabrikam123.com" This is possible because the Business456 pseudonym service interacts with
718 the Business456 IP and is authorized and allowed under the principal's privacy policy to reverse map
719 "XYZ321@Business456.com" into a known identity at Business456.com which has associated with it
720 pseudonyms for different realms. (Note that later in this section a mechanism for directly returning the
721 pseudonym by the IP is discussed). Figure 17 below illustrates this scenario:

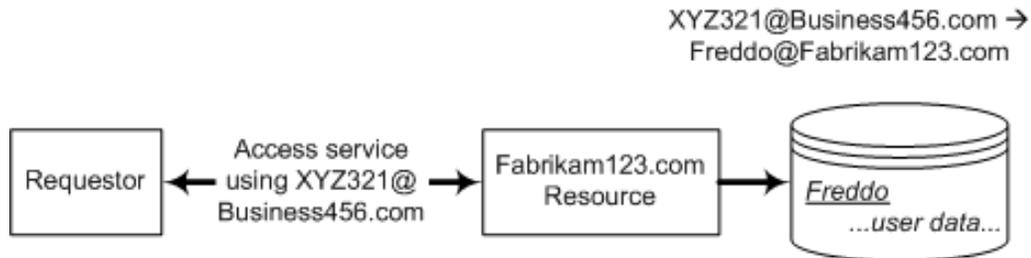


722

723

Figure 17: Pseudonym - local id

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

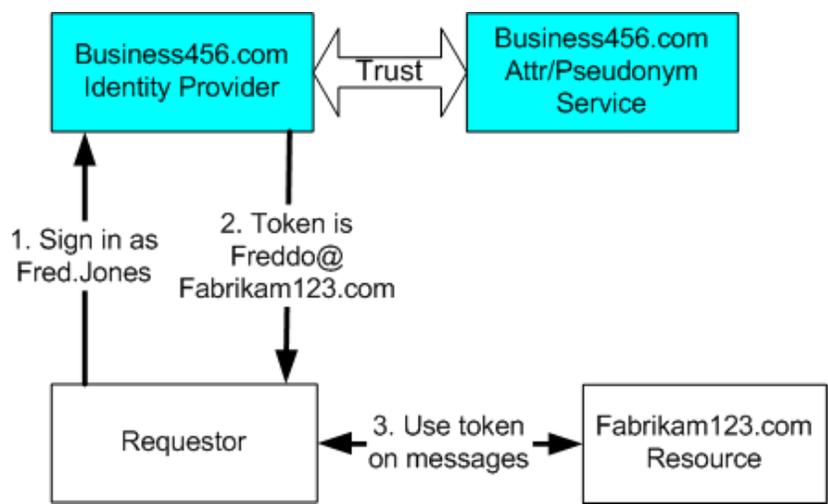


726

727

Figure 18: Pseudonym - local realm

728 Another variation of the first approach is to have the requestor map the identity, by creating pseudonyms
729 for specific services. In this case the Identity Provider (or STS) can operate hand-in-hand with the
730 pseudonym service. That is, the requestor asks its Identity Provider (or STS) for a token to a specified
731 trust realm or resource/service. The STS looks for pseudonyms and issues a token which can be used at
732 the specified resource/service as illustrated in figure 19 below:



733

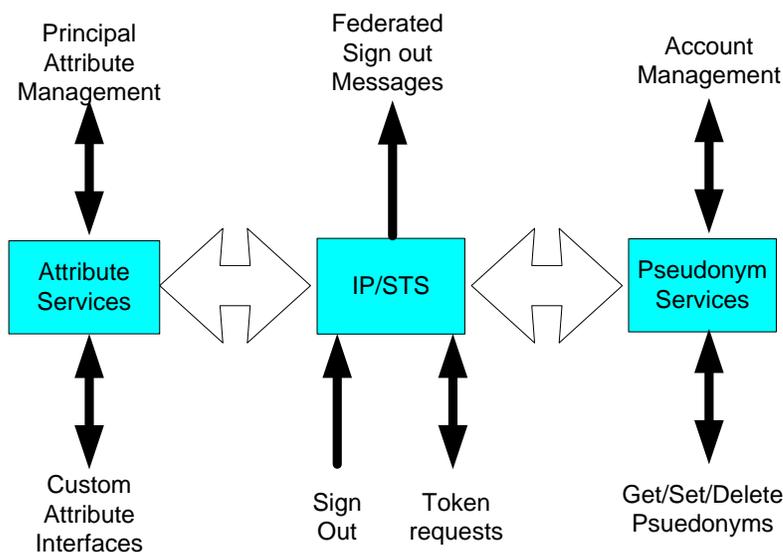
734

Figure 19: Pseudonym – token acceptance

735 The second approach is to create static identities for each service (or a group of services). That is,
 736 principle A at service X is given the digital identity 12, principle A at service Y is given the digital identity
 737 75, principle B at service X is given the digital identity 46, and so on. Operationally this approach is much
 738 like the last variation from the first approach. That is, the requestor must map its identity to an identity for
 739 the service (or service group) via a token request from its IP/STS (or using the pseudonym service
 740 directly). Consequently requestor mapping from random identities and pair-wise mapping are functionally
 741 equivalent.

742 2.7 Attributes, Pseudonyms, and IP/STS Services

743 This specification extends the WS-Trust model to allow attributes and pseudonyms to be integrated into
 744 the token issuance mechanism to provide federated identity mapping and attribute retrieval mechanisms,
 745 while protecting a principals' privacy. Any attribute, including pseudonyms, MAY be provided by an
 746 attribute or pseudonym service using the WS-Trust Security Token Service interface and token issuance
 747 protocol. Additional protocols or interfaces, especially for managing attributes and pseudonyms may
 748 MAY be supported; however, that is outside the scope of this specification. Figure 20 below illustrates the
 749 key aspects of this extended model:



750

751

Figure 20: Pseudonyms, Attributes and Token Issuance

752 As shown above, Principals request security tokens from Identity Providers and security token services.
753 As well, Principals MAY send sign-out requests (either explicitly as described later or implicitly by
754 cancelling tokens) indicating that cached or state information can be flushed immediately. Principals
755 request tokens for resources/service using the mechanisms described in WS-Trust and the issued tokens
756 may either represent the principals' primary identity or some pseudonym appropriate for the scope. The
757 Identity Provider (or STS) MAY send OPTIONAL sign-out notifications to subscribers (as described later).
758 Principals are associated with the attribute/pseudonym services and attributes and pseudonyms are
759 added and used.

3 Federation Metadata

760

761 Once two parties have made the decision to federate their computing systems, it is usually necessary to
762 configure their respective systems to enable federated operation. For example, the officers of a company
763 such as contoso.com might reach a business arrangement where they choose to provide a set of services
764 to someone who can present identity credentials (in the form of security tokens) issued by fabrikam.com.
765 In this example, it may be necessary for contoso.com administrator to update a local database with the
766 public key that fabrikam.com uses to sign its security tokens. In addition to the signing key, it may be
767 necessary for an organization to make available other types of information pertinent to a federated
768 relationship. Depending on the arrangement between the organizations, in some cases it is desirable to
769 help automate this configuration process.

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

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

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

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

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

3.1 Federation Metadata Document

787

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

791 The federation metadata document MUST be of the following form:

```
792 <?xml version="1.0" encoding="..." ?>  
793 <fed:FederationMetadata xmlns:fed="..." ...>  
794   <fed:Federation [FederationID="..."] ...> +  
795     [Federation Metadata]  
796   </fed:Federation>  
797   [Signature]  
798 </fed:FederationMetadata>
```

799 The document consists of one or more *federation* sections which describe the metadata for the endpoint
800 within a federation. The federation section MAY specify an URI indicating an identifier for the federation
801 using the `FederationID` attribute, or it MAY omit this identifier indicating the “default federation”. A

802 federation metadata document MUST NOT contain more than one default federation, that is, , only one
803 section may omit the FederationID attribute if multiple sections are provided.

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

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

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

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

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

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

819 **3.1.1 Referencing Other Metadata Documents**

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

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

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

```
837 <?xml version="1.0" encoding="utf-8" ?>
838 <fed:FederationMetadata xmlns:fed="..."
839     xmlns:mex="..."
840     xmlns:wsa="..."
841     xmlns:wsse="..."
842     xmlns:ds="...">
843   <fed:Federation>
844     <fed:TokenSigningKeyInfo>
845       <wsse:SecurityTokenReference>
846         <ds:X509Data>
847           <ds:X509Certificate>
848             ...
849           </ds:X509Certificate>
850         </ds:X509Data>
851       </wsse:SecurityTokenReference>
852     </fed:TokenSigningKeyInfo>
853     ...
854   </fed:Federation>
855   <fed:Federation FederationID="http://example.com/federation35532">
856     <mex:MetadataReference>
857       <wsa:Address>http://example.com/federation35332/FedMD
858     </wsa:Address>
859     </mex:MetadataReference>
860   </fed:Federation>
861   <fed:Federation FederationID="http://example.com/federation54478">
862     <mex:MetadataReference>
863       <wsa:Address>http://example.com/federation54478/FedMD
864     </wsa:Address>
865     </mex:MetadataReference>
866   </fed:Federation>
867 </fed:FederationMetadata>
```

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

874 For example, the following examples are functionally equivalent:

```
875 <?xml version="1.0" encoding="utf-8" ?>
876 <fed:FederationMetadata xmlns:fed="..." xmlns:wsse="..." xmlns:ds="...">
877   <fed:Federation FederationID="http://example.com/f1">
878     <fed:TokenSigningKeyInfo>
879       <wsse:SecurityTokenReference>
880         <ds:X509Data>
881           <ds:X509Certificate>
882             ...
883           </ds:X509Certificate>
884         </ds:X509Data>
885       </wsse:SecurityTokenReference>
886     </fed:TokenSigningKeyInfo>
887   </fed:Federation>
888   <fed:Federation FederationID="http://example.com/federation35532">
```

```

889     <fed:TokenSigningKeyInfo>
890         <wsse:SecurityTokenReference>
891             <ds:X509Data>
892                 <ds:X509Certificate>
893                     ...
894                 </ds:X509Certificate>
895             </ds:X509Data>
896         </wsse:SecurityTokenReference>
897     </fed:TokenSigningKeyInfo>
898 </fed:Federation>
899 </fed:FederationMetadata>

```

900 and

```

901 <?xml version="1.0" encoding="utf-8" ?>
902 <fed:FederationMetadata xmlns:fed="..." xmlns:wsse="..." xmlns:ds="...">
903     <fed:Federation FederationID="http://example.com/f1">
904         <fed:TokenSigningKeyInfo>
905             <wsse:SecurityTokenReference>
906                 <ds:X509Data>
907                     <ds:X509Certificate>
908                         ...
909                     </ds:X509Certificate>
910                 </ds:X509Data>
911             </wsse:SecurityTokenReference>
912         </fed:TokenSigningKeyInfo>
913     </fed:Federation>
914     <fed:Federation FederationID="http://example.com/federation35532">
915         <fed:FederationInclude>http://example.com/f1</fed:FederationInclude>
916     </fed:Federation>
917 </fed:FederationMetadata>

```

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

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

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

929 **3.1.2 TokenSigningKeyInfo Element**

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

```

936 <fed:TokenSigningKeyInfo ...>
937     <wsse:SecurityTokenReference>
938         ...
939     </wsse:SecurityTokenReference>
940 </fed:TokenSigningKeyInfo>

```

941

942 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
943 semantics defined in this specification.

944 For example, the token signing key can be carried inside an X.509 certificate and specified using the
945 ds:keyInfo element (as per [XMLDSIG]) as follows.

```
946 <fed:TokenSigningKeyInfo>  
947   <wsse:SecurityTokenReference>  
948     <ds:keyInfo>  
949       <ds:X509Data>  
950         <ds:X509Certificate>  
951 MIIBsTCCAV+gAwIBAgIQz9jmro9+5ahJyMQzgtSAvzAJBgUrDgMCHQUAMBYxFDASBgNVBAMTC1Jvb3  
952 QgQWdlbmN5MB4XDTA1MDkwMTEwNTUzNFoXDTM5MTIzMTIzNTk1OVowFDESMBAGA1UEAxMJbG9jYWxo  
953 b3N0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCnK1hCowhf6K3YrKoKuB87j6rdCrSHrnexzk  
954 Peg1YDwp6GquI3DVaD+VNRySREnI1yrqjDwyprp4FiJesPgs94PJRE6wz6Y5Z1CfhMUslh2t+XhBtJ  
955 ycvmlEZx+3lt2y6PCf49qlwfx/TqReCiMKYM9h+OVN32sFPQnz6dMUfH4QIDAQAB0swSTBHBgNVHQ  
956 EEQDA+gBAS5AktBh0dTWCNYSHcFmRjORgwFjEUMBIGA1UEAxMLUm9vdCBBZ2VuY3mCEAY3bACqAGSK  
957 Ec+41KpcNfQwCQYFKw4DAh0FAANBAFLkISG9ojZ2QtIfwJVJUdrsNzB08JZOrL181Zd9I//hZ6643  
958 L4sblBFB8ttbJjT4rdt5sKjpezRn3ZVIcvbQE=  
959       </ds:X509Certificate>  
960     </ds:X509Data>  
961   </ds:keyInfo>  
962 </wsse:SecurityTokenReference>  
963 </fed:TokenSigningKeyInfo>
```

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

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

```
968 <fed:TokenSigningKeyInfo>  
969   <wsse:SecurityTokenReference>  
970     <ds:RSAKeyValue>  
971       <ds:Modulus>  
972 A7SEU+e0yQH5rm9kbCDN9o3aPIo7HbP7tX6W0ocLZAtNfyxSZDU16ksL6WjubafOqNEpcwR3RdFsT7  
973 bCqnXPBe5ELh5u4VEy19MzxxXRgrMvavzyBpVRgBUU1V5foK5hhmbktQhyNdy/6LpQRhDUDsTvK+g  
974 9Ucj47es9AQJ3U=  
975       </ds:Modulus>  
976       <ds:Exponent>AQAB</ds:Exponent>  
977     </ds:RSAKeyValue>  
978   </wsse:SecurityTokenReference>  
979 </fed:TokenSigningKeyInfo>
```

980 3.1.3 TokenKeyTransferKeyInfo Element

981 The OPTIONAL <fed:TokenKeyTransferKeyInfo> element allows a federation metadata provider, a
982 security token service or Relying Party in this case, to specify what key should be used to encrypt keys
983 and key material targeted for the service. This is typically a service-level statement but can be an
984 endpoint-level statement. This element populates the [Federation Metadata] property. The key transfer
985 key can be specified using any of the mechanisms supported by the

986 <wsse:SecurityTokenReference> element defined in [WS-Security] as shown below.

```
987 <fed:TokenKeyTransferKeyInfo ...>  
988   <wsse:SecurityTokenReference>  
989     ...  
990   </wsse:SecurityTokenReference>  
991 </fed:TokenKeyTransferKeyInfo>
```

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

994 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
995 semantics defined in this specification.

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

```
997 <fed:TokenKeyTransferKeyInfo>  
998 <wsse:SecurityTokenReference>  
999 <ds:X509Data>  
1000 <ds:X509Certificate>  
1001 MIIBsTCCAV+gAwIBAgIQz9Jmro9+5ahJyMQzgtSAvzAJBgUrDgMCHQUAMBYxFDASBgNVBAMTC1Jvb3  
1002 QgQWdlbmN5MB4XDTA1MDkwMTEzNTk1OVowFDESMBAGA1UEAxMJbG9jYWx0  
1003 b3N0MIGfMA0GCSqGSIb3DQEBQUAA4GNADCBiQKBgQCNk1hCowhf6K3YrKoKuB87j6rdCrSHrnexzk  
1004 Peg1YDwp6GquI3DVaD+VNRySREnI1yrqjDWyprp4FiJesPgs94PJRE6wz6Y5Z1CfhMUslh2t+XhBtJ  
1005 ycvmlEZx+3Lt2y6PCf49qlwFX/TqReCiMKYM9h+OVN32sFPQnz6dMUfH4QIDAQABo0swSTBHBgNVHQ  
1006 EEQDA+gBAS5AktBh0dTWCNYSHcFmRjoRgwFjEUMBIGAlUEAxMLUm9vdCBBZ2VuY3mCEAY3bACqAGSK  
1007 Ec+41KpcNfQwCQYFKw4DAh0FAANBAFLkIsG9ojZ2QtIfwjVJUdrsNzBO8JZOrLl81Zd9I//hZ6643  
1008 L4sblBFB8ttbJjT4rdt5sKjpezRn3ZVIcvbQE=  
1009 </ds:X509Certificate>  
1010 </ds:X509Data>  
1011 </wsse:SecurityTokenReference>  
1012 </fed:TokenSigningKeyInfo>
```

1013 Note that if this element isn't specified, and the signing key doesn't prohibit key transfer, it MAY be used
1014 as the key transfer key.

1015 3.1.4 IssuerNamesOffered Element

1016 In some scenarios token issuers are referred to be a logical name representing an equivalence class of
1017 issuers. For example, a Relying Party may not care what specific bank issues a token so long as the
1018 issuance is associated with a specific credit card program. To facilitate this, federated metadata provides
1019 the `<sp:TokenIssuerName>` element (described in [WS-SecurityPolicy]) to indicate that a Relying Party
1020 needs a token from a specific class of issuer.

1021 As stated, the OPTIONAL `<fed:IssuerNamesOffered>` element allows a federation metadata
1022 provider, specifically a token service in this case, to specify a set of "logical names" that are associated
1023 with the provider. That is, when a Relying Party indicates a logical name for a token issuer using the
1024 `<sp:TokenIssuerName>` element in a token assertion the `<fed:IssuerNamesOffered>` element
1025 this element can be used as a correlation mechanism by clients. This element populates the [Federation
1026 Metadata] property. This is typically a service-level statement but can be an endpoint-level statement.

1027 The schema for this optional element is shown below.

```
1028 <fed:IssuerNamesOffered ...>  
1029 <fed:IssuerName Uri="xs:anyURI" .../> +  
1030 </fed:IssuerNamesOffered>
```

1031 The following example illustrates using this optional element to specify a logical name of the federating
1032 organization as a token issuer.

```
1033 <fed:IssuerNamesOffered>  
1034 <fed:IssuerName Uri="http://fabrikam.com/federation/corporate" />  
1035 </fed:IssuerNamesOffered>
```

1036 3.1.5 TokenIssuerEndpoints Element

1037 The OPTIONAL `<fed:TokenIssuerEndpoints>` element allows a federation metadata provider to
1038 specify the endpoint address of a trusted STS (or addresses of functionally equivalent STSs) which can

1039 be referenced by federated partners when requesting tokens from it. This element populates the
1040 [Federation Metadata] property. This is specified by token issuers and security token services. This is
1041 typically a service-level statement but can be an endpoint-level statement. The schema for this optional
1042 element is shown below.

```
1043 <fed:TokenIssuerEndpoints>  
1044   wsa:EndpointReferenceType +  
1045 </fed:TokenIssuerEndpoints>
```

1046 The content of this element is one, or more, endpoint references as defined by [WS-Addressing]
1047 providing a transport address for the issuer STS(or functionally equivalent STS endpoints). Each
1048 endpoint reference MAY (and SHOULD if there is no expectation that the policy is known *a priori*) include
1049 metadata for the STS endpoint or a reference to an endpoint from where such metadata can be retrieved
1050 by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for additional details).

1051 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1052 semantics defined in this specification.

1053 It should be noted that this element MAY occur multiple times indicating distinct services with different
1054 capabilities. Service providers MUST include functionally equivalent endpoints – different endpoint
1055 references for a single service, or for a set of logically equivalent services – in a single
1056 <fed:TokenIssuerEndpoints> element.

1057 The following example illustrates using this optional element to specify an endpoint address for the token
1058 issuing STS of the federating organization.

```
1059 <fed:TokenIssuerEndpoints>  
1060   <wsa:Address> http://fabrkam.com/federation/STS </wsa:Address>  
1061 </fed:TokenIssuerEndpoints>
```

1062

1063 3.1.6 TokenIssuerMetadata Element

1064 The optional <fed:TokenIssuerMetadata> element allows a federation metadata provider to specify the
1065 metadata corresponding to its token issuing service (or addresses for functionally equivalent security
1066 token services) which can be referenced by federated partners when requesting tokens from it. This
1067 element populates the [Federation Metadata] property. This is specified by token issuers and security
1068 token services. This is a service-level statement.

1069 The schema for this optional element is shown below.

```
1070 <fed:TokenIssuerMetadata>  
1071   <mex:Metadata> ... </mex:metadata>  
1072 </fed:TokenIssuerMetadata>
```

1073 The content of this element is Metadata element as defined by [WS-MetadataExchange] providing a
1074 representation of the metadata for the issuer STS (or functionally equivalent STS endpoints).

1075 This element allows attributes to be added so long as they do not alter the semantics defined in this
1076 specification.

1077 The following example illustrates using this optional element to specify a metadata address for the token
1078 issuing STS of an organization. This address may be used to look up the endpoint address for the STS.

```
1079 <fed:TokenIssuerMetadata>  
1080   <mex:Metadata>  
1081     <mex:MetadataSection Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">  
1082       <wsx:MetadataReference>  
1083         <wsa:Address>  
1084           https://fabrikam.com/identityserver/trust/mex  
1085         </wsa:Address>  
1086       </wsx:MetadataReference>
```

1087
1088
1089

```
</mex:MetadataSection>  
</mex:Metadata>  
</fed:TokenIssuerMetadata>
```

1090

1091 **3.1.7 PseudonymServiceEndpoints Element**

1092 The OPTIONAL `<fed:PseudonymServiceEndpoints>` element allows a federation metadata provider
1093 to specify the endpoint address of its pseudonym service (or addresses for functionally equivalent
1094 pseudonym services) which can be referenced by federated partners when requesting tokens from it.
1095 When present, this indicates that services SHOULD use the pseudonym service to map identities to local
1096 names as the identities MAY vary across invocations. This element populates the [Federation Metadata]
1097 property. This is typically specified by token issuers and security token services. This is typically a
1098 service-level statement but can be an endpoint-level statement.

1099 The schema for this optional element is shown below.

1100
1101
1102

```
<fed:PseudonymServiceEndpoints>  
  wsa:EndpointReferenceType +  
</fed:PseudonymServiceEndpoints>
```

1103 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1104 a transport address for an STS interface to the pseudonym service (or functionally equivalent pseudonym
1105 service endpoints). Each endpoint reference MAY (and SHOULD if there is no expectation that the policy
1106 is known *a priori*) include metadata for the STS endpoint or a reference to an endpoint from where such
1107 metadata can be retrieved by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for
1108 additional details).

1109 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1110 semantics defined in this specification.

1111 It should be noted that this element MAY occur multiple times indicating distinct services with different
1112 capabilities. Service providers MUST include equivalent endpoints – different endpoint references for a
1113 single service, or for a set of logically equivalent services – in a single
1114 `<fed:PseudonymServiceEndpoints>` element.

1115 The following example illustrates using this optional element to specify an endpoint address for the
1116 pseudonym service of the federating organization.

1117
1118
1119

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

1120 **3.1.8 AttributeServiceEndpoints Element**

1121 The OPTIONAL `<fed:AttributeServiceEndpoints>` element allows a federation metadata
1122 provider to specify the endpoint address of its attribute service (or addresses for functionally equivalent
1123 attribute services) which can be referenced by federated partners when requesting tokens from it. This
1124 element populates the [Federation Metadata] property. This is typically specified by requestors and is a
1125 service-level statement.

1126 The schema for this optional element is shown below.

1127
1128
1129

```
<fed:AttributeServiceEndpoints>  
  wsa:EndpointReferenceType +  
</fed:AttributeServiceEndpoints>
```

1130 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1131 a transport address for an STS interface to the service (or functionally equivalent attribute service
1132 endpoints). Each endpoint reference MAY (and SHOULD if there is no expectation that the policy is
1133 known *a priori*) include metadata for the STS endpoint or a reference to an endpoint from where such
1134 metadata can be retrieved by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for
1135 additional details).

1136 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1137 semantics defined in this specification.

1138 It should be noted that this element MAY occur multiple times indicating distinct services with different
1139 capabilities. Service providers MUST include equivalent endpoints – different endpoint references for a
1140 single service, or for a set of logically equivalent services – in a single <fed:AttributeServiceEndpoints>
1141 element.

1142 The following example illustrates using this optional element to specify an endpoint address for the
1143 attribute service of the federating organization.

```
1144 <fed:AttributeServiceEndpoints>  
1145   <wsa:Address> http://fabrkam.com/federation/Attr </wsa:Address>  
1146 </fed:AttributeServiceEndpoints>
```

1147 **3.1.9 SingleSignOutSubscriptionEndpoints Element**

1148 The OPTIONAL <fed:SingleSignOutSubscriptionEndpoints> element allows a federation
1149 metadata provider to specify the endpoint address of its subscription service (or addresses for functionally
1150 equivalent subscription services) which can be used to subscribe to federated sign-out messages. This
1151 element populates the [Federation Metadata] property. This is typically specified by token issuers and
1152 security token services. This is typically a service-level statement but can be an endpoint-level statement.

1153 The schema for this optional element is shown below.

```
1154 <fed:SingleSignOutSubscriptionEndpoints>  
1155   wsa:EndpointReferenceType +  
1156 </fed:SingleSignOutSubscriptionEndpoints>
```

1157 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1158 a transport address for the subscription manager (or functionally equivalent subscription services).

1159 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1160 semantics defined in this specification.

1161 **3.1.10 SingleSignOutNotificationEndpoints Element**

1162 Services MAY subscribe for sign-out notifications however clients MAY also push notifications to services.

1163 The OPTIONAL <fed:SingleSignOutNotificationEndpoints> element allows a federation
1164 metadata provider to specify the endpoint address (or functionally equivalent addresses) to which push
1165 notifications of sign-out are to be sent. This element populates the [Federation Metadata] property. This
1166 is typically specified by service providers and security token services. This is typically a service-level
1167 statement but can be an endpoint-level statement.

1168 The schema for this optional element is shown below.

```
1169 <fed:SingleSignOutNotificationEndpoints>  
1170   wsa:EndpointReferenceType +  
1171 </fed:SingleSignOutNotificationEndpoints>
```

1172 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1173 a transport address for the notification service (or functionally equivalent notification service endpoints) .

1174 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1175 semantics defined in this specification.

1176 3.1.11 TokenTypesOffered Element

1177 The OPTIONAL `<fed:TokenTypesOffered>` element allows a federation metadata provider to specify
1178 the list of offered security token types that can be issued by its STS. A federated partner can use the
1179 offered token types to decide what token type to ask for when requesting tokens from it. This element
1180 populates the [Federation Metadata] property. This is typically specified by token issuers and security
1181 token services. This is typically a service-level statement but can be an endpoint-level statement.

1182 The schema for this optional element is shown below.

```
1183 <fed:TokenTypesOffered ...>  
1184   <fed:TokenType Uri="xs:anyURI" ...>  
1185     ...  
1186   </fed:TokenType> +  
1187   ...  
1188 </fed:TokenTypesOffered>
```

1189 The following describes the elements listed in the schema outlined above:

1190 `/fed:TokenTypesOffered`

1191 This element is used to express the list of token types that the federating STS is capable of
1192 issuing.

1193 `/fed:TokenTypesOffered/fed:TokenType`

1194 This element indicates an individual token type that the STS can issue.

1195 `/fed:TokenTypesOffered/fed:TokenType/@Uri`

1196 This attribute provides the unique identifier (URI) of the individual token type that the STS can
1197 issue.

1198 `/fed:TokenTypesOffered/fed:TokenType/{any}`

1199 The semantics of any content for this element are undefined. Any extensibility or use of sub-
1200 elements MUST NOT alter the semantics defined in this specification.

1201 `/fed:TokenTypesOffered/fed:TokenType/@{any}`

1202 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
1203 MUST NOT violate or alter the semantics defined in this specification.

1204 `/fed:TokenTypesOffered/@{any}`

1205 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
1206 MUST NOT violate or alter the semantics defined in this specification.

1207 `/fed:TokenTypesOffered/{any}`

1208 The semantics of any content for this element are undefined. Any extensibility or use of sub-
1209 elements MUST NOT alter the semantics defined in this specification.

1210 The following example illustrates using this optional element to specify that the issuing STS of the
1211 federating organization can issue both SAML 1.1 and SAML 2.0 tokens [WSS:SAMLTokenProfile].

```
1212 <fed:TokenTypesOffered>  
1213   <fed:TokenType Uri="urn:oasis:names:tc:SAML:1.1" />  
1214   <fed:TokenType Uri="urn:oasis:names:tc:SAML:2.0" />  
1215 </fed:TokenTypesOffered>
```

1216 3.1.12 ClaimTypesOffered Element

1217 The OPTIONAL `<fed:ClaimTypesOffered>` element allows a federation metadata provider such as
1218 an IdP to specify the list of publicly offered claim types, named using the schema provided by the
1219 common claims dialect defined in this specification, that can be asserted in security tokens issued by its
1220 STS. It is out of scope of this specification whether or not a URI used to name a claim type resolves.
1221 Note that issuers MAY support additional claims and that not all claims may be available for all token
1222 types. If other means of describing/identifying claims are used in the future, then corresponding XML
1223 elements can be introduced to publish the new claim types. A federated partner can use the offered claim
1224 types to decide which claims to ask for when requesting tokens from it. This specification places no
1225 requirements on the syntax used to describe the claims. This element populates the [Federation
1226 Metadata] property. This is typically specified by token issuers and security token services. This is
1227 typically a service-level statement but can be an endpoint-level statement.

1228 The schema for this optional element is shown below.

```
1229 <fed:ClaimTypesOffered ...>  
1230   <auth:ClaimType ...> ... </auth:ClaimType> +  
1231 </fed:ClaimTypesOffered>
```

1232 The following describes the elements listed in the schema outlined above:

1233 `/fed:ClaimTypesOffered`

1234 This element is used to express the list of claim types that the STS is capable of issuing.

1235 `/fed:ClaimTypesOffered/@{any}`

1236 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
1237 NOT alter the semantics defined in this specification.

1238 The following example illustrates using this optional element to specify that the issuing STS of the
1239 federating organization can assert two claim types named using the common claims format.

```
1240 <fed:ClaimTypesOffered>  
1241   <auth:ClaimType Uri="http://.../claims/EmailAddr" >  
1242     <auth:DisplayName>Email Address</auth:DisplayName>  
1243   </auth:ClaimType>  
1244   <auth:ClaimType Uri="http://.../claims/IsMember" >  
1245     <auth:DisplayName>Is a Member (yes/no)</auth:DisplayName>  
1246     <auth:Description>If a person is a member of this club</auth:Description>  
1247   </auth:ClaimType>  
1248 </fed:ClaimTypesOffered>
```

1249

1250 3.1.13 ClaimDialectsOffered Element

1251 The OPTIONAL `fed:ClaimDialectsOffered` element allows a federation metadata provider to specify the
1252 list of dialects, named using URIs, that are accepted by its STS in token requests to express the claims
1253 requirement. A federated partner can use is list to decide which dialect to use to express its desired
1254 claims when requesting tokens from it. This specification defines one standard claims dialect in the
1255 subsequent section 9.3, but other claim dialects MAY be defined elsewhere for use in other scenarios.
1256 This element populates the [Federation Metadata] property. This is typically specified by token issuers
1257 and security token services. This is typically a service-level statement but can be an endpoint-level
1258 statement.

1259 The schema for this optional element is shown below.

```
1260 <fed:ClaimDialectsOffered>  
1261   <fed:ClaimDialect Uri="xs:anyURI" /> +  
1262 </fed:ClaimDialectsOffered>
```

1263 The following describes the elements listed in the schema outlined above:

1264 `/fed:ClaimDialectsOffered`

1265 This element is used to express the list of claim dialects that the federating STS can understand
1266 and accept.

1267 `/fed:ClaimDialectsOffered/fed:ClaimDialect`

1268 This element indicates an individual claim dialect that the STS can understand.

1269 `/fed:ClaimDialectsOffered/fed:ClaimDialect/@Uri`

1270 This attribute provides the unique identifier (URI) of the individual claim dialect that the STS can
1271 understand.

1272 `/fed:ClaimDialectsOffered/fed:ClaimDialect/...`

1273 The semantics of any content for this element are undefined. Any extensibility or use of sub-
1274 elements MUST NOT alter the semantics defined in this specification.

1275 `/fed:ClaimDialectsOffered/fed:ClaimDialect/@{any}`

1276 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
1277 MUST NOT violate or alter the semantics defined in this specification.

1278 `/fed:ClaimDialectsOffered/@{any}`

1279 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
1280 MUST NOT violate or alter the semantics defined in this specification.

1281 The following example illustrates using this optional element to specify that the issuing STS of the
1282 federating organization can accept the one standard claims dialect defined in this specification.

1283

```
1284 <fed:ClaimDialectsOffered>
1285   <fed:ClaimDialect Uri="http://schemas.xmlsoap.org/ws/2005/05/fedclaims" />
1286 </fed:ClaimDialectsOffered>
```

1287 **3.1.14 AutomaticPseudonyms Element**

1288 The OPTIONAL `<fed:AutomaticPseudonyms>` element allows a federation metadata provider to
1289 indicate if it automatically maps pseudonyms or applies some form of identity mapping. This element
1290 populates the [Federation Metadata] property. This is typically specified by token issuers and security
1291 token services. This is typically a service-level statement but can be an endpoint-level statement. If not
1292 specified, requestors SHOULD assume that the service does not perform automatic mapping (although it
1293 MAY).

1294 The schema for this optional element is shown below.

```
1295 <fed:AutomaticPseudonyms>
1296   xs:boolean
1297 </fed:AutomaticPseudonyms>
```

1298 **3.1.15 PassiveRequestorEndpoints Element**

1299 The optional `<fed:PassiveRequestorEndpoints>` element allows a federation metadata provider,
1300 security token service, or relying party to specify the endpoint address that supports the Web (Passive)
1301 Requestor protocol described below in section 13. This element populates the [Federation Metadata]
1302 property. This is an endpoint-level statement.

1303 The schema for this optional element is shown below.

1304
1305
1306

```
<fed:PassiveRequestorEndpoints>  
  <wsa:EndpointReference> ... </wsa:EndpointReference>+  
</fed:PassiveRequestorEndpoints>
```

1307 The content of this element is an endpoint reference element as defined by [WS-Addressing] that
1308 identifies an endpoint address that supports receiving the Web (Passive) Requestor protocol messages
1309 described below in section 13.
1310 This element allows attributes to be added so long as they do not alter the semantics defined in this
1311 specification.

1312 It should be noted that this element MAY occur multiple times indicating distinct endpoints with different
1313 capabilities. Service providers MUST include functionally equivalent endpoints in a single
1314 <fed:PassiveRequestorEndpoints> element.

1315 The following example illustrates using this optional element to specify the endpoint address that supports
1316 the Web (Passive) Requestor protocol described in section 13 for the token issuing STS of the federating
1317 organization.

```
1318 <fed:PassiveRequestorEndpoints>  
1319   <wsa:EndpointReference>  
1320     <wsa:Address> http://fabrikam.com/federation/STS/Passive </wsa:Address>  
1321   </wsa:EndpointReference>  
1322 </fed:PassiveRequestorEndpoints>  
1323
```

1324 3.1.16 TargetScopes Element

1325 The [WS-Trust] protocol allows a token requester to indicate the target where the issued token will be
1326 used (i.e., token scope) by using the optional element `wsp:AppliesTo` in the RST message. To
1327 communicate the supported `wsp:AppliesTo` (`wtrealm` values in passive requestor scenarios) for a realm,
1328 federated metadata provides the `<fed:TargetScopes>` element to indicate the EPRs that are associated
1329 with token scopes of the relying party or STS. Note that an RP or STS MAY be capable of supporting
1330 other `wsp:AppliesTo` values. This element populates the [Federation Metadata] property. This is typically
1331 a service-level statement.

1332 The schema for this optional element is shown below.

1333
1334
1335
1336
1337

```
<fed:TargetScopes ...>  
  <wsa:EndpointReference>  
    ...  
  </wsa:EndpointReference> +  
</fed:TargetScopes>
```

1338 The following example illustrates using this optional element to specify a logical name of the federating
1339 organization as a token issuer.

1340
1341
1342
1343
1344

```
<fed:TargetScopes >  
  <wsa:EndpointReference>  
    <wsa:Address> http://fabrikam.com/federation/corporate </wsa:Address>  
  </wsa:EndpointReference>  
</fed:TargetScopes >
```

1345

1346 3.1.17 ContactInfoAddress Element

1347 The OPTIONAL <fed:ContactInfoAddresses> element allows a federation metadata provider to specify
1348 the endpoint addresses to be used for contacting the metadata provider for further details on the services
1349 and capabilities described in the metadata. This element populates the [Federation Metadata] property.

1350 The schema for this optional element is shown below.

1351

```
1352 <fed:ContactInfoAddresses>  
1353 ( <fed:WebPage> xs:anyURI </fed:WebPage> *  
1354 <fed:Email> xs:anyURI </fed:Email> * ) +  
1355 ...  
1356 </fed:ContactInfoAddresses> ?
```

1357

1358 /fed:ContactInfoAddresses

1359 The content of this OPTIONAL element is one or more elements that provide references to web
1360 pages with contact info about the federation services and/or an email address to use as a contact
1361 point.

1362 This element allows other attributes to be added so long as they do not alter the semantics
1363 defined in this specification.

1364 /fed: ContactInfoAddresses/fed:WebPage

1365 This element of type xs:anyURI MAY appear 0 or more times, it's content should be a valid
1366 [HTTP] scheme URI that resolves to a web page with contact information regarding the federation
1367 services and/or metadata document.

1368 /fed: ContactInfoAddresses/fed:Email

1369 This element of type xs:anyURI MAY appear 0 or more times, it's content should be a valid
1370 [mailto] scheme URI regarding the federation services and/or metadata document.

1371 3.1.18 [Signature] Property

1372 The OPTIONAL [Signature] property provides a digital signature over the federation metadata document
1373 to ensure data integrity and provide data origin authentication. The provider of a federation metadata
1374 document SHOULD include a digital signature over the metadata document, and consumers of the
1375 metadata document SHOULD perform signature verification if a signature is present.

1376 The token used to sign this document MUST speak for the endpoint. If the metadata is for a token issuer
1377 then the key used to sign issued tokens SHOULD be used to sign this document. This means that if a
1378 <fed:TokenSigningKey> is specified, it SHOULD be used to sign this document.

1379 This section describes the use of [XML-Signature] to sign the federation metadata document, but other
1380 forms of digital signatures MAY be used for the [Signature] property. XML Signature is the
1381 RECOMMENDED signing mechanism. The [Signature] property (in the case of XML Signature this is
1382 represented by the <ds:Signature> element) provides the ability for a federation metadata provider
1383 organization to sign the metadata document such that a partner organization consuming the metadata
1384 can authenticate its origin.

1385 The signature over the federation metadata document MUST be signed using an enveloped signature
1386 format as defined by the [XML-Signature] specification. In such cases the root of the signature envelope
1387 MUST be the <fed:FederationMetadata> element as shown in the following example. If the
1388 metadata document is included inside another XML document, such as a SOAP message, the root of the
1389 signature envelope MUST remain the same. Additionally, XML Exclusive Canonicalization [XML-C14N]
1390 MUST be used when signing with [XML-Signature].

```

1391 (01) [<?xml version='1.0' encoding=... > ]
1392 (02) <fed:FederationMetadata
1393 (03)   xmlns:fed="..." xmlns:ds="..."
1394 (04)   wsu:Id="_fedMetadata">
1395 (05)   ...
1396 (06)   <ds:Signature xmlns:ds="...">
1397 (07)     <ds:SignedInfo>
1398 (08)       <ds:CanonicalizationMethod Algorithm="..." />
1399 (09)       <ds:SignatureMethod Algorithm="..." />
1400 (10)       <ds:Reference URI="_fedMetadata">
1401 (11)         <ds:Transforms>
1402 (12)           <ds:Transform Algorithm=".../xmldsig#enveloped-signature" />
1403 (13)           <ds:Transform Algorithm=".../xml-exc-c14n#" />
1404 (14)         </ds:Transforms>
1405 (15)         <ds:DigestMethod Algorithm="..." />
1406 (16)         <ds:DigestValue>xdJRPBPERvaZD9gTt4e6Mg==</ds:DigestValue>
1407 (17)       </ds:Reference>
1408 (18)     </ds:SignedInfo>
1409 (19)     <ds:SignatureValue> mpcFEK6JuUFBPojQ8VBW2Q==</ds:SignatureValue>
1410 (20)     <ds:KeyInfo>
1411 (21)       ...
1412 (22)     </ds:KeyInfo>
1413 (23)   </ds:Signature>
1414 (24) </fed:FederationMetadata>

```

1415 Note that the enveloped signature contains a single `ds:Reference` element (line 10) containing a URI
1416 reference to the `<fed:FederationMetadata>` root element (line 04) of the metadata document.
1417

1418 3.1.19 Example Federation Metadata Document

1419 The following example illustrates a signed federation metadata document that uses the OPTIONAL
1420 metadata elements described above and an enveloped [XML Signature] to sign the document.

```

1421 <?xml version="1.0" encoding="utf-8" ?>
1422 <fed:FederationMetadata wsu:Id="_fedMetadata"
1423   xmlns:fed="..." xmlns:wsu="..." xmlns:wsse="..." xmlns:ds="..."
1424   xmlns:wsa="...">
1425 <fed:Federation>
1426   <fed:TokenSigningKeyInfo>
1427     <wsse:SecurityTokenReference>
1428       <ds:X509Data>
1429         <ds:X509Certificate>
1430           MIIBsTCCA+g...zRn3ZVIcvbQE=
1431         </ds:X509Certificate>
1432       </ds:X509Data>
1433     </wsse:SecurityTokenReference>
1434   </fed:TokenSigningKeyInfo>
1435   <fed:TokenIssuerName>
1436     http://fabrikam.com/federation/corporate
1437   </fed:TokenIssuerName>
1438   <fed:TokenIssuerEndpoint>
1439     <wsa:Address> http://fabrkam.com/federation/STS </wsa:Address>
1440   </fed:TokenIssuerEndpoint>
1441   <fed:TokenTypesOffered>
1442     <fed:TokenType Uri="urn:oasis:names:tc:SAML:1.1" />
1443     <fed:TokenType Uri="urn:oasis:names:tc:SAML:2.0" />
1444   </fed:TokenTypesOffered>
1445
1446   <fed:ClaimTypesOffered>
1447     <auth:ClaimType Uri="http://.../claims/EmailAddr" >
1448     <auth:DisplayName>Email Address</auth:DisplayName>

```

```

1449     </auth:ClaimType>
1450     <auth:ClaimType Uri="http://.../claims/IsMember" >
1451       <auth:DisplayName>Is a Member (yes/no)</auth:DisplayName>
1452       <auth:Description>If a person is a member of this club</auth:Description>
1453     </auth:ClaimType>
1454   </fed:ClaimTypesOffered> </fed:Federation>
1455
1456   <ds:Signature xmlns:ds="...">
1457     <ds:SignedInfo>
1458       <ds:CanonicalizationMethod Algorithm="..." />
1459       <ds:SignatureMethod Algorithm="..." />
1460       <ds:Reference URI="_fedMetadata">
1461         <ds:Transforms>
1462           <ds:Transform Algorithm=".../xmldsig#enveloped-signature" />
1463           <ds:Transform Algorithm=".../xml-exc-c14n#" />
1464         </ds:Transforms>
1465         <ds:DigestMethod Algorithm="..." />
1466         <ds:DigestValue>xdJRPBPERvaZD9gTt4e6Mg==</ds:DigestValue>
1467       </ds:Reference>
1468     </ds:SignedInfo>
1469     <ds:SignatureValue>mpcFEK6JuUFBPoJQ8VBW2Q==</ds:SignatureValue>
1470     <ds:KeyInfo>
1471       ...
1472     </ds:KeyInfo>
1473   </ds:Signature>
1474 </fed:FederationMetadata>

```

1475 3.2 Acquiring the Federation Metadata Document

1476 This section provides specific details and restrictions on how a party may securely obtain the federation
 1477 metadata document for a *target domain* representing a target organization it wishes to federate with. It
 1478 should be noted that some providers of federation metadata documents MAY require authentication of
 1479 requestors or MAY provide different (subset) documents if requestors are not authenticated.

1480 It is assumed that the target domain is expressed as a fully-qualified domain name (FQDN). In other
 1481 words, it is expressed as the DNS domain name of the target organization, e.g., fabrikam.com.

1482 It should be noted that compliant services are NOT REQUIRED to support all of the mechanisms defined
 1483 in this section. If a client only has a DNS host name and wants to obtain the federation metadata, the
 1484 following order is the RECOMMENDED bootstrap search order:

- 1485 1. Use the well-known HTTPS address with the federation ID
- 1486 2. Use the well-known HTTPS address for the default federation
- 1487 3. Use the well-known HTTP address with the federation ID
- 1488 4. Use the well-known HTTP address for the default federation
- 1489 5. Look for any DNS SRV records indicating federation metadata locations

1490 If multiple locations are available and no additional prioritization is specified, the following order is the
 1491 RECOMMENDED download processing order:

- 1492 1. HTTPS
- 1493 2. WS-Transfer/WS-ResourceTransfer
- 1494 3. HTTP

1495 **3.2.1 WSDL**

1496 The metadata document MAY be included within a WSDL document using the extensibility mechanisms
1497 of WSDL. Specifically the `<fed:FederationMetadata>` element can be placed inside of WSDL
1498 documents in the same manner as policy documents are as specified in WS-PolicyAttachment.
1499 The metadata document can appear in WSDL for a service, port, or binding.

1500 **3.2.2 The Federation Metadata Path**

1501 A default path MAY be supported to provide federation metadata. The path for obtaining the federation
1502 metadata document for the default federation for a target domain denoted by ***target-DNS-domain***
1503 SHOULD be constructed as follows:

1504 `http://server-name/FederationMetadata/spec-version/FederationMetadata.xml`

1505 or

1506 `https://server-name/FederationMetadata/spec-version/FederationMetadata.xml`

1507 where

1508 *server-name* is the host name (DNS name) of a server providing the federation metadata document. It
1509 SHOULD be obtained by doing a DNS query of SRV records for ***target-DNS-domain*** as
1510 described in Section 3.2.6. If no DNS record is found, then the target DNS domain name MUST
1511 BE used as the default value of the server name as well.

1512 *spec-version* is the version of the federation metadata specification supported by the acquiring party. For
1513 this version of the specification the ***spec-version*** MUST BE the string "2007-06".

1514 Implementations MAY choose to use a short form of the target DNS domain name, such as the primary
1515 domain and suffix, but this choice is implementation specific.

1516 The following subsections describe the mechanisms through which the federation metadata document for
1517 a target domain may be acquired by a federating party. The target domain MUST support at least one of
1518 the mechanisms described below, but MAY choose to support more than one mechanism.

1519 It is RECOMMENDED that a target domain (or organization) that makes federation metadata available for
1520 acquisition by partners SHOULD publish DNS SRV resource records to allow an acquiring party to locate
1521 the servers where the metadata is available. The type and format of the SRV resource records to be
1522 published in DNS is described in Section 3.2.6. These records correspond to each metadata acquisition
1523 mechanism specified in the following subsections.

1524 If a specific federation context is known, the following URLs SHOULD be checked prior to checking for
1525 the default federation context.

1526 `http://server-name/FederationMetadata/spec-version/fed-id/FederationMetadata.xml`

1527 or

1528 `https://server-name/FederationMetadata/spec-version/fed-id/FederationMetadata.xml`

1529 where

1530 *fed-id* is the `FederationID` value described previously for identifying a specific federation.

1531 **3.2.3 Retrieval Mechanisms**

1532 The following OPTIONAL retrieval mechanisms are defined:

1533 **Using HTTP**

1534 The federation metadata document may be obtained from the following URL using HTTP GET
1535 mechanism:

1536

```
http:path
```

1537 where *path* is constructed as described in Section 3.2.2.

1538 Metadata signatures are RECOMMENDED when using HTTP download.

1539 Using HTTPS

1540 The federation metadata document MAY be obtained from the following URL using HTTPS GET
1541 mechanism:

1542

```
https:path
```

1543 where *path* is constructed as described in Section 3.2.2.

1544 There is no requirement that the HTTPS server key be related to the signing key identified in the
1545 metadata document, but it is RECOMMENDED that requestors verify that both keys can speak for the
1546 target service.

1547 Using WS-Transfer/WS-ResourceTransfer

1548 The federation metadata document can be obtained by sending the [WS-Transfer] "Get" operation to an
1549 endpoint that serves that metadata as described in [WS-MetadataExchange] (see also section 3.2.5).

1550 Note that the [WS-ResourceTransfer] extensions MAY be used to filter the metadata information returned.

1551 The use of [WS-Security] with [WS-Transfer/WS-ResourceTransfer] is RECOMMENDED to authenticate
1552 the sender and protect the integrity of the message.

1553 3.2.4 FederatedMetadataHandler Header

1554 If an endpoint reference for metadata obtained via SOAP requests is not already available to a requester
1555 (e.g. when only a URL is know), the requestor SHOULD include the

1556 <fed:FederationMetadataHandler> header to allow metadata requests to be quickly identified.

1557 The syntax is as follows:

1558

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

1559 The<fed:FederationMetadataHandler> header SHOULD NOT use a S:mustUnderstand='1'
1560 attribute. Inclusion of this header allows a front-end service to know that federation metadata is being
1561 requested and perform header-based routing.

1562 The following example illustrates a [WS-Transfer] with [WS-ResourceTransfer] extensions request
1563 message to obtain the federation metadata document for an organization with contoso.com as its domain
1564 name.

1565

```
1566 (01) <s12:Envelope  
1567 (02)   xmlns:s12="..."  
1568 (03)   xmlns:wsa="..."  
1569 (04)   xmlns:wsxf="..."  
1570 (05)   xmlns:fed="...">  
1571 (06) <s12:Header>  
1572 (07)   <wsa:Action>  
1573 (08)     http://schemas.xmlsoap.org/ws/2004/09/transfer/Get  
1574 (09)   </wsa:Action>  
1575 (10)   <wsa:MessageID>  
1576 (11)     uuid:73d7edfd-5c3d-b949-46ba-02decaee433f  
1577 (12)   </wsa:MessageID>  
1578 (13)   <wsa:ReplyTo>  
1579 (14)     <wsa:Address>http://fabrikam.com/Endpoint</wsa:Address>  
1580 (15)   </wsa:ReplyTo>  
1581 (16)   <wsa:To>  
1582 (17)     http://contoso.com/FederationMetadata/2007-06/FederationMetadata.xml  
1583 (18)   </wsa:To>
```

```
1583 (19) <fed:FederatedMetadataHandler />
1584 (20) </s12:Header>
1585 (21) <s12:Body />
1586 (22) </s12:Envelope>
```

1587 The response to the [WS-Transfer] with [WS-ResourceTransfer] extensions request message is illustrated
1588 below.

```
1589 (01) <s12:Envelope
1590 (02)   xmlns:s12="..."
1591 (03)   xmlns:wsa="..."
1592 (04)   xmlns:wsxf="..."
1593 (05)   xmlns:fed="...">
1594 (06) <s12:Header>
1595 (07)   <wsa:To>http://fabrikam.com/Endpoint</wsa:To>
1596 (08)   <wsa:Action>
1597 (09)     http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
1598 (10)   </wsa:Action>
1599 (11)   <wsa:MessageID>
1600 (12)     uuid:86d7eac5-6e3d-b869-64bc-35edacee743d
1601 (13)   </wsa:MessageID>
1602 (14)   <wsa:RelatesTo>
1603 (15)     uuid:73d7edfd-5c3d-b949-46ba-02decaee433f
1604 (16)   </wsa:RelatesTo>
1605 (17) </s12:Header>
1606 (18) <s12:Body>
1607 (19)   <fed:FederationMetadata
1608 (20)     xmlns:fed="...">
1609 (21)     ...
1610 (22)   </fed:FederationMetadata>
1611 (21) </s12:Body>
1612 (22) </s12:Envelope>
```

1613 3.2.5 Metadata Exchange Dialect

1614 The federation metadata document MAY be included as a metadata unit within a Web service
1615 <mex:Metadata> element, which is a collection of metadata units, using the metadata unit inclusion
1616 mechanisms described in [WS-MetadataExchange]. This can be done by including a
1617 <mex:MetadataSection> element that contains the federation metadata document in-line or by
1618 reference. To facilitate inclusion of the federation metadata as a particular type of metadata unit, the
1619 following metadata dialect URI is defined in this specification that MUST be used as the value of the
1620 <mex:MetadataSection/@Dialect> XML attribute:

```
1621 http://docs.oasis-open.org/wsfed/federation/200706
```

1622 No identifiers for federation metadata units, as specified by the value of the OPTIONAL
1623 <mex:MetadataSection/@Identifier> XML attribute, are defined in this specification.

1624 For example, a federation metadata unit specified in-line within a <mex:Metadata> element is shown
1625 below:

```
1626 <mex:Metadata>
1627   <mex:MetadataSection
1628     Dialect='http://docs.oasis-open.org/wsfed/federation/200706'>
1629     <fed:FederationMetadata ...>
1630     ...
1631   </fed:FederationMetadata>
```

1632 <mex:MetadataSection>

1633 <mex:Metadata>

1634 3.2.6 Publishing Federation Metadata Location

1635 A target domain (or organization) that makes federation metadata available for acquisition by partners
1636 SHOULD publish SRV resource records in the DNS database to allow an acquiring party to locate the
1637 servers where the metadata is available. The specific format and content of the SRV resource records to
1638 be published is described here.

1639 The SRV record is used to map the name of a service (in this case the federation metadata service) to
1640 the DNS hostname of a server that offers the service. For more information about SRV resource records,
1641 see [DNS-SRV-RR]. The general form of the *owner name* of a SRV record to be published is as follows:

1642 `_Service.Protocol.TargetDnsDomain`

1643 In this case, a target domain offers the “federation metadata” service over one or more of the protocol
1644 mechanisms described earlier (namely, HTTP, HTTPS or WS-Transfer/WS-ResourceTransfer). For each
1645 protocol mechanism supported by a target domain, a corresponding SRV record SHOULD be published in
1646 DNS as follows.

1647 If acquisition of the federation metadata document using HTTP GET (Section 3.2.3) is supported, then the
1648 owner name of the published SRV record MUST be of the form below:

1649 `_fedMetadata._http.TargetDnsDomain`

1650 If acquisition of the federation metadata document using HTTPS GET (Section 3.2.3) is supported, then
1651 the owner name of the published SRV record MUST be of the form below:

1652 `_fedMetadata._https.TargetDnsDomain`

1653 If acquisition of the federation metadata document using [WS-Transfer/WS-ResourceTransfer] (Section
1654 3.2.3) is supported, then the owner name of the published SRV record MUST be of the form below:

1655 `_fedMetadata._wsxfr._http.TargetDnsDomain`

1656 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.

1657 Note that if multiple protocols are specified with the same priority, the requestor MAY use any protocol or
1658 process in any order it chooses.

1659 The following example illustrates the complete SRV records published by the organization with domain
1660 name “contoso.com” that makes its federation metadata available over all three mechanisms discussed
1661 earlier.

1662

```
1663 server1.contoso.com IN A 128.128.128.0  
1664 server2.contoso.com IN A 128.128.128.1  
1665 _fedMetadata._http.contoso.com IN SRV 0 0 80 server1.contoso.com  
1666 _fedMetadata._https.contoso.com IN SRV 0 0 443 server1.contoso.com
```

1667

```
_fedMetadata._wsxfr.contoso.com IN SRV 0 0 80 server2.contoso.com
```

1668 A client attempting to acquire the federation metadata for a target domain using any selected protocol
1669 mechanism SHOULD query DNS for SRV records using one of the appropriate owner name forms
1670 described above.

1671 **3.2.7 Federation Metadata Acquisition Security**

1672 It is RECOMMENDED that a target domain publishing federation metadata SHOULD include a signature
1673 in the metadata document using a key that is authorized to "speak for" the target domain. If the metadata
1674 contains a `<fed:TokenSigningKey>` element then this key SHOULD be used for the signature. If
1675 there are multiple `Federation` elements specified then the default scope's signing key SHOULD be
1676 used. If there is no default scope then the choice is up to the signer. Recipients of federation metadata
1677 SHOULD validate that signature to authenticate the metadata publisher and verify the integrity of the
1678 data. Specifically, a recipient SHOULD verify that the key used to sign the document has the right to
1679 "speak for" the target domain (see *target-DNS-domain* in Section 3.2.2) with which the recipient is trying
1680 to federate. See also the security considerations at the end of this document.

1681 4 Sign-Out

1682 The purpose of a *federated sign-out* is to clean up any cached state and security tokens that may exist
1683 within the federation, but which are no longer required. In typical usage, sign-out notification serves as a
1684 hint – upon termination of a principal's session – that it is OK to flush cached data (such as security
1685 tokens) or state information for that specific principal. It should be noted that a sign-out message is a
1686 *one-way* message. No "sign-out-complete" reply message can be required since the Sign-Out operation
1687 cannot be guaranteed to complete. Further, sign-out requests might be processed in batch, causing a
1688 time delay that is too long for the request and response to be meaningfully correlated. In addition,
1689 requiring a Web browser requestor to wait for a successful completion response could introduce arbitrary
1690 and lengthy delays in the user experience. The processing implication of sign-out messages can vary
1691 depending on the type of application that is being used to sign-out. For example, the implication of sign-
1692 out on currently active transactions is undefined and is resource-specific.

1693 In some cases, formal sign-out is implicit or not required. This section defines messages that MAY be
1694 used by profiles for explicit sign-out.

1695 In general, sign-out messages are unreliable and correct operation must be ensured in their absence (i.e.,
1696 the messages serve as hints only). Consequently, these messages MUST also be treated as idempotent
1697 since multiple deliveries could occur.

1698 When sign-out is supported, it is typically provided as part of the IP/STS as it is usually the central
1699 processing point.

1700 Sign-out is separate from token cancellation as it applies to all tokens and all target sites for the principal
1701 within the domain/realm.

1702 4.1 Sign-Out Message

1703 The sign-out mechanism allows requestors to send a message to its IP/STS indicating that the requester
1704 is initiating a termination of the SSO. That is, cached information or state information can safely be
1705 flushed. This specification defines OPTIONAL sign-out messages that MAY be used. It should be noted,
1706 however, that the typical usage pattern is that only token issuance and message security are used and
1707 sign-out messages are only for special scenarios. Sign-out messages, whether from the client to the
1708 IP/STS, from the IP/STS to a subscriber, or from the client to a service provider, all use the same
1709 message form described in this section.

1710 For SOAP, the action of this message is as follows:

1711 `http://docs.oasis-open.org/wsfed/federation/200706/SignOut`

1712 The following represents an overview of the syntax of the `<fed:SignOut>` element:

```
1713 <fed:SignOut wsu:Id="..." ...>  
1714   <fed:Realm>xs:anyURI</fed:Realm> ?  
1715   <fed:SignOutBasis ...>...<fed:SignOutBasis>  
1716   ...  
1717 </fed:SignOut>
```

1718 The following describes elements and attributes used in a `<fed:SignOut>` element.

1719 `/fed:SignOut`

1720 This element represents a sign-out message.

1721 `/fed:SignOut/fed:Realm`

1722 This OPTIONAL element specifies the "realm" to which the sign-out applies and is specified as a
1723 URI. If no realm is specified, then it is assumed that the recipient understands and uses a
1724 fixed/default realm.

1725 /fed:SignOut/fed:SignOutBasis

1726 The contents of this REQUIRED element indicate the principal that is signing out. Note that any
 1727 security token or security token reference MAY be used here and multiple tokens MAY be
 1728 specified. That said, it is expected that the <UsernameToken> will be the most common. Note
 1729 that a security token or security token reference MUST be specified.

1730 /fed:SignOut/fed:SignOutBasis/@{any}

1731 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added
 1732 to the element. Use of this extensibility mechanism MUST NOT alter the semantics of this
 1733 specification.

1734 /fed:SignOut/fed:SignOutBasis/{any}

1735 This is an extensibility mechanism to allow the inclusion of the relevant security token reference
 1736 or security token(s).

1737 /fed:SignOut/@wsu:Id

1738 This OPTIONAL attribute specifies a string label for this element.

1739 /fed:SignOut/@{any}

1740 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added
 1741 to the element. Use of this extensibility mechanism MUST NOT alter the semantics of this
 1742 specification.

1743 /fed:SignOut/{any}

1744 This is an extensibility mechanism to allow additional elements to be used. For example, an STS
 1745 might use extensibility to further qualify the sign-out basis. Use of this extensibility mechanism
 1746 MUST NOT alter the semantics of this specification.

1747

1748 The <fed:SignOut> message SHOULD be signed by the requestor to prevent tampering and to
 1749 prevent unauthorized sign-out messages (i.e., Alice sending a sign-out message for Bob without Bob's
 1750 knowledge or permission). The signature SHOULD contain a timestamp to prevent replay attacks (see
 1751 WS-Security for further discussion on this). It should be noted, however, that a principal MAY delegate
 1752 the right to issue such messages on their behalf. The following represents an example of the
 1753 <fed:SignOut> message:

```

1754 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
1755   xmlns:wsu="..." xmlns:wsse="...">
1756   <S:Header>
1757     ...
1758     <wsu:Timestamp wsu:Id="ts">
1759       ...
1760     </wsu:Timestamp>
1761     <wsse:Security>
1762       <!-- Signature referecing IDs "ts" & "so" -->
1763       ...
1764     </wsse:Security>
1765   </S:Header>
1766   <S:Body>
1767     <fed:SignOut wsu:Id="so">
1768       <fed:SignOutBasis>
1769         <wsse:UsernameToken>
1770           <wsse:Username>NNK</wsse:Username>
1771         </wsse:UsernameToken>
1772       </fed:SignOutBasis>
1773     </fed:SignOut>
1774   </S:Body>
1775 </S:Envelope>
  
```

1776 4.2 Federating Sign-Out Messages

1777 In many environments there is a need to take the messages indicating sign-out and distribute them
1778 across the federation, subject to authorization and privacy rules. Consequently, these messages result
1779 from when an explicit message is sent to the IP/STS (by either the principal or a delegate such as an
1780 IP/STS), or implicitly from an IP/STS as a result of some other action (such as a token request).

1781 In the typical use case, federated sign-out messages will be generated by the principal terminating a
1782 session, either at the “primary STS” (the IP/STS that manages the principal’s identity) or at one of the
1783 resource providers (or its STS) accessed during the session. There are two primary flows for these
1784 messages. In one case they are effectively chained through all the STSs involved in the session; that is,
1785 a mechanism is used (if available) by the “primary STS” to send sign-out messages to all the other STSs
1786 in a sequential manner by causing each message to cause the next message to occur in sequence
1787 resulting in a message back to itself either on completion or at each step to orchestrate the process. The
1788 second approach is to require the “primary STS” to send sign-out messages to all the other token
1789 services and target services in parallel (those that it knows about).

1790 The chained (sequential) approach has been found to be fragile. If one of the message fails to complete
1791 its local processing and does not pass the sign-out message on – or the network partitions – the sign-out
1792 notification does not reach all the involved parties. For this reason, compliant implementations SHOULD
1793 employ the parallel approach. If the session is terminated at a resource provider, it SHOULD clean up
1794 any local state and then send a sign-out message to the “primary STS”. The latter SHOULD send parallel
1795 sign-out messages to all the other STSs.

1796 Sessions MAY involve secondary branches (between token services at different resources) of which the
1797 “primary STS” has no knowledge. In these cases, the appropriate resource token services SHOULD
1798 perform the role of “primary STS” for sign-out of these branches.

1799 It should be noted that clients MAY also push (send) sign-out messages directly to other services such as
1800 secondary IP/STSs or service providers.

1801 Sign-out could potentially be applied to one of two different scopes for the principal’s session. Sign-out
1802 initiated at the “primary STS” SHOULD have global scope and apply to all resource STSs and all
1803 branches of the session. Sign-out initiated at a resource STS could also have global scope as described
1804 above. However, it could also be considered as a request to clean up only the session state related to
1805 that particular resource provider. Thus implementations MAY provide a mechanism to restrict the scope
1806 of federated sign-out requests that originate at a resource STS to its particular branch of the principal’s
1807 session. This SHOULD result in cleaning up all state at (or centered upon) that STS. It SHOULD involve
1808 a request to be sent to the “primary STS” to clean up session state only for that particular STS or
1809 resource provider.

1810 Federated sign-out request processing could involve providing status messages to the user. This
1811 behavior is implementation specific and out-of-scope of this specification.

1812 The result of a successful request is that all compliant SSO messages generated implicitly or explicitly are
1813 sent to the requesting endpoints if allowed by the authorization/privacy rules.

1814 SSO messages MAY be obtained by subscribing to the subscription endpoint using the mechanisms
1815 described in [WS-Eventing]. The subscription endpoint, if available, is described in the federation
1816 metadata document.

1817 The [WS-Eventing] mechanisms allow for subscriptions to be created, renewed, and cancelled. SSO
1818 subscriptions MAY be filtered using the XPath filter defined in [WS-Eventing] or using the SSO filter
1819 specified by the following URI:

1820 `http://docs.oasis-open.org/wsfed/federation/200706/ssoevt`

1821 This filter allows the specification of a realm and security tokens to restrict the SSO messages. The
1822 syntax is as follows:

```
1823 <wse:Subscribe ...>
1824   ...
1825   <wse:Filter Dialect=".../federation/ssoevt">
1826     <fed:Realm>...</fed:Realm> ?
1827     ...security tokens...
1828   </wse:Filter>
1829   ...
1830 </wse:Subscribe>
```

1831 The following describes elements and attributes illustrated above:

1832 /wse:Filter/fed:Realm

1833 This OPTIONAL element specifies the "realm" to which the sign-out applies. At most one
1834 <fed:Realm> can be specified. The contents of this element are the same type and usage as in
1835 the *fed:Signout/fed:Realm* described above. If this element is not specified it is assumed
1836 that either the subscription service knows how to infer the correct realm and uses a single
1837 service-determined realm or the request fails. Note that if multiple realms are desired then
1838 multiple subscriptions are needed.

1839 /wse:Filter/... security tokens(s) ...

1840 The contents of these OPTIONAL elements restrict messages to only the specified identities.
1841 Note that any security token or security token reference MAY be used here and multiple tokens
1842 MAY be specified. That said, it is expected that the <wsse:UsernameToken> will be the most
1843 common. Note that if multiple tokens are specified they represent a logical OR – that is,
1844 messages that match any of the tokens for the corresponding realm are reported.

1845 This filter dialect does not allow any contents other than those described above. If no filter is specified
1846 then the subscription service MAY fail or MAY choose a default filter for the subscription.

5 Attribute Service

1847

1848 Web services often need to be able to obtain additional data related to service requestors to provide the
1849 requestor with a richer (e.g. personalized) experience. This MAY be addressed by having an attribute
1850 service that requesters and services MAY use to access this additional information. In many cases, the
1851 release of this information about a service requestor is subject to authorization and privacy rules and
1852 access to this data (or the separate service that has data available for such purposes) is only granted to
1853 authorized services for any given attribute.

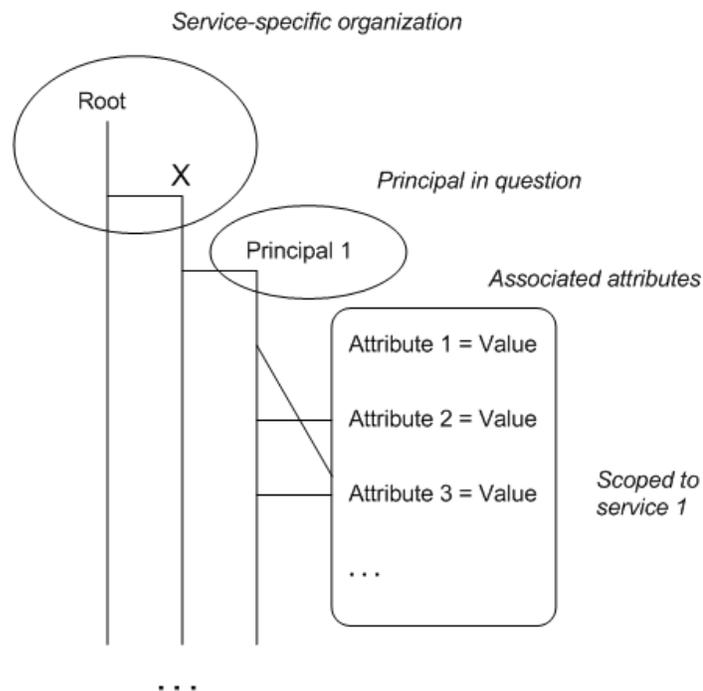
1854 Attribute stores most likely exist in some form already in service environments using service-specific
1855 protocols (e.g. such as LDAP). An attribute service provides the interface to this attribute store.

1856 Figure 21 below illustrates the conceptual namespace of an attribute service.

1857 An attribute service MAY leverage existing repositories and may MAY provide some level of organization
1858 or context. That is, this specification makes no proposals or requirements on the organization of the data,
1859 just that if a principal exists, any corresponding attribute data should be addressable using the
1860 mechanisms described here.

1861 Principals represent any kind of resource, not just people. Consequently, the attribute mechanisms MAY
1862 be used to associate attributes with any resource, not just with identities. Said another way, principal
1863 identities represent just one class of resource that can be used by this specification.

1864 Principals and resources MAY have specific policies that are required when accessing and managing
1865 their attributes. Such policies use the [WS-Policy] framework. As well, these principals (and resources)
1866 MAY be specified as domain expressions to scope policy assertions as described in [WS-
1867 PolicyAttachment].



1868

1869

Figure 21 Attribute Service

1870 It is expected that separate attributes MAY be shared differently and MAY require different degrees of
1871 privacy and protection. Consequently, each attribute expression SHOULD be capable of expressing its
1872 own access control and privacy policy. As well, the access control and privacy policy SHOULD take into
1873 account the associated scope(s) and principals that can speak for the scope(s).

1874 Different services MAY support different types of attribute services which MAY be identified via policy by
1875 definition of new policy assertions indicating the attribute service supported.

1876 Each attribute store MAY support different subsets of the functionality as described above. The store's
1877 policy indicates what functionality it supports.

1878 This specification does not require a specific attribute service definition or interface. However, as
1879 indicated in sections 2.7 and 3.1.8, the WS-Trust Security Token Service interface and token issuance
1880 protocol MAY be used as the interface to an attribute service. Reusing an established service model and
1881 protocol could simplify threat analysis and implementation of attribute services.

1882

6 Pseudonym Service

1883

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

1884

1885

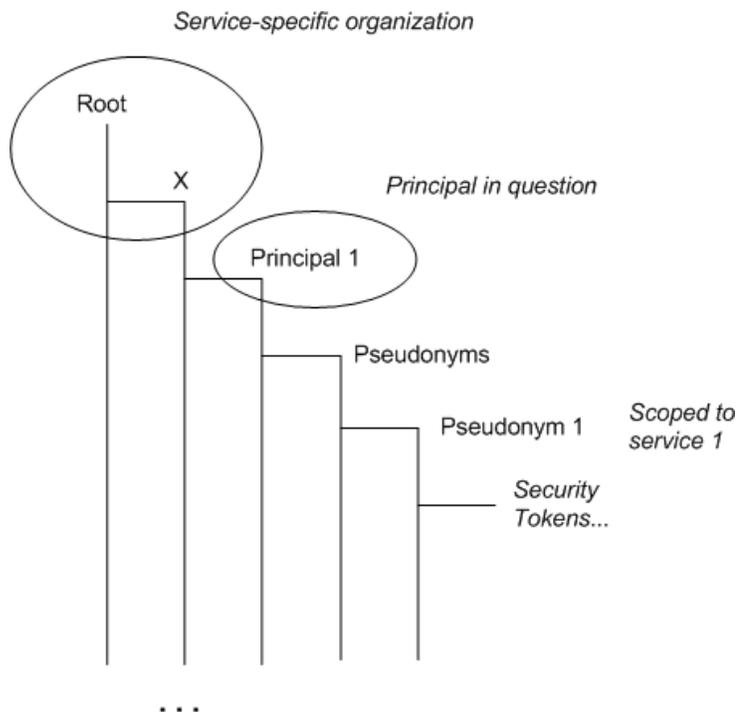
Pseudonym services MAY exist in some form already in service environments using service-specific protocols. This specification defines an additional, generic, interface to these services for interoperability with Web services.

1886

1887

1888

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



1889

1890

Figure 22 Pseudonym Service

1891

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.

1892

1893

1894

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).

1895

1896

1897

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.

1898

1899

1900

1901

1902

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).

1903

1904

1905

1906

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

1907

1908 6.1 Filtering Pseudonyms

1909 When performing operations on a pseudonym store it is RECOMMENDED to filter the scope of the
1910 operation. This is done using the following dialect with the [WS-ResourceTransfer] extensions to [WS-
1911 Transfer]:

1912 `http://docs.oasis-open.org/wsfed/federation/200706/pseudonymdialect`

1913 Alternatively, the <fed:FilterPseudonyms> header MAY be specified with WS-Transfer to allow
1914 filtering to be specified as part of an endpoint reference (EPR).

1915 The syntax for the <fed:FilterPseudonyms> element is as follows:

```
1916 <fed:FilterPseudonyms ...>  
1917   <fed:PseudonymBasis ...>...</fed:PseudonymBasis> ?  
1918   <fed:RelativeTo ...>...</fed:RelativeTo> ?  
1919   ...  
1920 </fed:FilterPseudonyms>
```

1921 The following describes elements and attributes used in a <fed:FilterPseudonyms> element.

1922 /fed:FilterPseudonyms

1923 This element indicates a request to filter a pseudonym operation based on given identity
1924 information and applicability scope.

1925 /fed:FilterPseudonyms/fed:PseudonymBasis

1926 This element specifies a security token or security token reference identifying the known identity
1927 information. This element is typically required to identify the basis but MAY be omitted if the
1928 context is known. This specification places no requirements on what information (claims) are
1929 required to be a pseudonym basis – that can vary by service.

1930 /fed:FilterPseudonyms/fed:PseudonymBasis/@{any}

1931 This is an extensibility point allowing attributes to be specified. Use of this extensibility
1932 mechanism MUST NOT alter semantics defined in this specification.

1933 /fed:FilterPseudonyms/fed:PseudonymBasis/{any}

1934 This is an extensibility mechanism to allow the inclusion of the relevant security token reference
1935 or security token.

1936 /fed:FilterPseudonyms/fed:RelativeTo

1937 This RECOMMENDED element indicates the scope for which the pseudonym is requested. This
1938 element has the same type as <wsp:AppliesTo>.

1939 /fed:FilterPseudonyms/fed:RelativeTo/@{any}

1940 This is an extensibility point allowing attributes to be specified.

1941 Use of this extensibility mechanism MUST NOT alter the semantics of this specification.

1942 alter semantics defined in this specification.

1943 /fed:FilterPseudonyms/@{any}

1944 This is an extensibility point allowing attributes to be specified. Use of this extensibility
1945 mechanism MUST NOT . alter semantics defined in this specification.

1946 /fed:FilterPseudonyms/{any}

1947 This is an extensibility point allowing content elements to be specified.

1948 Use of this extensibility mechanism MUST NOT alter semantics defined in this specification.

1949 As noted above, in some circumstances it MAY be desirable to include a filter as part of an EPR. To
1950 accommodate this, <fed:FilterPseudonyms> element MAY be specified as a SOAP header. It is
1951 RECOMMENDED that the SOAP *mustUnderstand* attribute be specified as *true* whenever this is used as
1952 a header. If a <fed:FilterPseudonyms> header is specified, the message MUST NOT contain
1953 additional filtering.

1954 6.2 Getting Pseudonyms

1955 Pseudonyms are requested from a pseudonym service using the [WS-Transfer] “GET” method with the
1956 [WS-ResourceTransfer] extensions. The dialect defined in 6.1 (or the <fed:FilterPseudonyms>
1957 header) is used to restrict the pseudonyms that are returned.

1958 Pseudonyms are returned in the body of the GET response message in a <fed:Pseudonym> element
1959 as follows:

```
1960 <fed:Pseudonym ...>  
1961   <fed:PseudonymBasis ...>...</fed:PseudonymBasis>  
1962   <fed:RelativeTo ...>...</fed:RelativeTo>  
1963   <wsu:Expires>...</wsu:Expires>  
1964   <fed:SecurityToken ...>...</fed:SecurityToken> *  
1965   <fed:ProofToken ...>...</fed:ProofToken> *  
1966   ...  
1967 </fed:Pseudonym>
```

1968 The following describes elements and attributes described above:

1969 /fed:Pseudonym

1970 This element represents a pseudonym for a principal.

1971 /fed:Pseudonym/fed:PseudonymBasis

1972 This element specifies a security token or security token reference identifying the known identity
1973 information (see [WS-Security]). Often this is equivalent to the basis in the request although if
1974 multiple pseudonyms are returned that value may be different.

1975 /fed:Pseudonym/fed:PseudonymBasis/@{any}

1976 This is an extensibility point allowing attributes to be specified.

1977 Use of this extensibility mechanism MUST NOT alter semantics defined in this specification.

1978 /fed:Pseudonym/fed:PseudonymBasis/{any}

1979 This is an extensibility mechanism to allow the inclusion of the relevant security token reference
1980 or security token. Use of this extensibility mechanism MUST NOT alter semantics defined in this
1981 specification.

1982 /fed:Pseudonym/fed:RelativeTo

1983 This REQUIRED element indicates the scope for which the pseudonym is requested. This
1984 element has the same type as <wsp:AppliesTo>.

1985 /fed:Pseudonym/fed:RelativeTo/@{any}

1986 This is an extensibility point allowing attributes to be specified. Use of this extensibility
1987 mechanism MUST NOT alter semantics defined in this specification.

1988 /fed:Pseudonym/wsu:Expires

1989 This OPTIONAL element indicates the expiration of the pseudonym.

1990 /fed:Pseudonym/fed:SecurityToken

1991 This OPTIONAL element indicates a security token for the scope. Note that multiple tokens MAY
1992 be specified.

- 1993 /fed:Pseudonym/fed:SecurityToken/@{any}
- 1994 This is an extensibility point allowing attributes to be specified. Use of this extensibility
1995 mechanism MUST NOT alter semantic defined in this specification.
- 1996 /fed:Pseudonym/fed:SecurityToken/{any}
- 1997 This is an extensibility mechanism to allow the inclusion of the relevant security token(s). Use of
1998 this extensibility mechanism MUST NOT alter semantics defined in this specification
- 1999 /fed:Pseudonym/fed:ProofToken
- 2000 This OPTIONAL element indicates a proof token for the scope. Note that multiple tokens MAY be
2001 specified.
- 2002 /fed:Pseudonym/fed:ProofToken/@{any}
- 2003 This is an extensibility point allowing attributes to be specified. Use of this extensibility
2004 mechanism MUST NOT alter semantics defined in this specification.
- 2005 /fed:Pseudonym/fed:ProofToken/{any}
- 2006 This is an extensibility mechanism to allow the inclusion of the relevant security token(s). Use of
2007 this extensibility mechanism MUST NOT alter semantics defined in this specification.
- 2008 /fed:Pseudonym/@{any}
- 2009 This is an extensibility point allowing attributes to be specified. Use of this extensibility
2010 mechanism MUST NOT alter semantics defined in this specification.
- 2011 /fed:Pseudonym/{any}
- 2012 This is an extensibility point allowing content elements to be specified. Use of this extensibility
2013 mechanism MUST NOT alter semantics defined in this specification.
- 2014 For example, the following example obtains the local pseudonym associated with the identity (indicated
2015 binary security token) for the locality (target scope) indicated by the URI
2016 <http://www.fabrikam123.com/NNK>.

```

2017 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
2018   xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
2019   <S:Body>
2020     <wsrt:Get
2021       Dialect="http://docs.oasis-open.org/wsrf/federation/200706/pseudonymdialect">
2022       <wsrt:Expression>
2023         <fed:FilterPseudonyms>
2024           <fed:PseudonymBasis>
2025             <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2026           </fed:PseudonymBasis>
2027           <fed:RelativeTo>
2028             <wsa:Address>
2029               http://www.fabrikam123.com/NNK
2030             </wsa:Address>
2031           </fed:RelativeTo>
2032         </fed:FilterPseudonyms>
2033       </wsrt:Expression>
2034     </wsrt:Get>
2035   </S:Body>
2036 </S:Envelope>

```

2037 A sample response might be as follows:

```

2038 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
2039   xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
2040   <S:Body>
2041     <wsrt:GetResponse>
2042       <wsrt:Result>

```

```

2043     <fed:Pseudonym>
2044         <fed:RelativeTo>
2045             <wsa:Address>
2046                 http://www.fabrikam123.com/NNK
2047             </wsa:Address>
2048         </fed:RelativeTo>
2049         <wsu:Expires>2003-12-10T09:00Z</wsu:Expires>
2050         <fed:SecurityToken>...</fed:SecurityToken>
2051         <fed:ProofToken>...</fed:ProofToken>
2052     </fed:Pseudonym>
2053 </wsrt:Result>
2054 </wsrt:GetResponse>
2055 </S:Body>
2056 </S:Envelope>

```

2057 6.3 Setting Pseudonyms

2058 Pseudonyms are updated in a pseudonym service using the [WS-Transfer] “PUT” operation with the [WS-
2059 ResourceTransfer] extensions using the dialect defined in 6.1 (or the <fed:FilterPseudonyms>
2060 header). This allows one or more pseudonyms to be added. If a filter is not specified, then the PUT
2061 impacts the full pseudonym set. It is RECOMMENDED that filters be used.

2062 The following example sets pseudonym associated with the identity (indicated binary security token) for
2063 the locality (target scope) indicated by the URI <http://www.fabrikam123.com/NNK>.

```

2064 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
2065     xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
2066 <S:Body>
2067 <wsrt:Put
2068     Dialect="http://docs.oasis-open.org/wsrf/federation/200706/pseudonymdialect">
2069 <wsrt:Fragment Mode="Inset">
2070 <wsrt:Expression>
2071 <fed:FilterPseudonyms>
2072 <fed:PseudonymBasis>
2073 <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2074 </fed:PseudonymBasis>
2075 <fed:RelativeTo>
2076 <wsa:Address>
2077     http://www.fabrikam123.com/NNK
2078 </wsa:Address>
2079 </fed:RelativeTo>
2080 </fed:FilterPseudonyms>
2081 </wsrt:Expression>
2082 <wsrt:Value>
2083 <fed:Pseudonym>
2084 <fed:PseudonymBasis>
2085 <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2086 </fed:PseudonymBasis>
2087 <fed:RelativeTo>
2088 <wsa:Address>
2089     http://www.fabrikam123.com/NNK
2090 </wsa:Address>
2091 </fed:RelativeTo>
2092 <fed:SecurityToken>
2093 <wsse:UsernameToken>
2094 <wsse:Username> "Nick" </wsse:Username>
2095 </wsse:UsernameToken>
2096 </fed:SecurityToken>
2097 <fed:ProofToken>...</fed:ProofToken>
2098 </fed:Pseudonym>
2099 </wsrt:Value>
2100 </wsrt:Fragment>

```

2101
2102
2103

```
</wsrt:Put>  
</S:Body>  
</S:Envelope>
```

2104 6.4 Deleting Pseudonyms

2105 Pseudonyms are deleted in a pseudonym service using the [WS-Transfer] “PUT” operation with the [WS-
2106 ResourceTransfer] extensions. The dialect defined in 6.1 (or the <fed:FilterPseudonyms> header) is
2107 used to restrict the scope of the “PUT” to only remove pseudonym information corresponding to the filter.
2108 If a filter is not specified, then the PUT impacts the full pseudonym set. It is RECOMMENDED that filters
2109 be used.

2110 The following example deletes the pseudonym associated with the identity (indicated binary security
2111 token) for the locality (target scope) indicated by the URI <http://www.fabrikam123.com/NNK>.

2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."  
  xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">  
  <S:Body>  
    <wsrt:Put  
      Dialect="http://docs.oasis-open.org/wsfed/federation/200706/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>
```

2134 6.5 Creating Pseudonyms

2135 Pseudonyms are created in a pseudonym service using the WS-Resource “CREATE” operation with the
2136 [WS-ResourceTransfer] extensions. This allows one or more pseudonyms to be added. The dialect
2137 defined in 6.1 (or the <fed:FilterPseudonyms> header) is specified on the CREATE to only create
2138 pseudonym information corresponding to the filter. If a filter is not specified, then the CREATE impacts
2139 the full pseudonym set. It is RECOMMENDED that filters be used.

2140 The following example creates pseudonym associated with the identity (indicated binary security token)
2141 for the locality (target scope) indicated by the URI <http://www.fabrikam123.com/NNK>.

2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."  
  xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">  
  <S:Body>  
    <wsrt:Create  
      Dialect="http://docs.oasis-open.org/wsfed/federation/200706/pseudonymdialect">  
        <wsrt:Fragment>  
          <wsrt:Expression>  
            <fed:FilterPseudonyms>  
              <fed:PseudonymBasis>  
                <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>  
              </fed:PseudonymBasis>  
              <fed:RelativeTo>
```

```
2154         <wsa:Address>
2155             http://www.fabrikam123.com/NNK
2156         </wsa:Address>
2157     </fed:RelativeTo>
2158 </fed:FilterPseudonyms>
2159 </wsrt:Expression>
2160 <wsrt:Value>
2161     <fed:Pseudonym>
2162         <fed:PseudonymBasis>
2163             <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2164         </fed:PseudonymBasis>
2165         <fed:RelativeTo>
2166             <wsa:Address>
2167                 http://www.fabrikam123.com/NNK
2168             </wsa:Address>
2169         </fed:RelativeTo>
2170         <fed:SecurityToken>
2171             <wsse:UsernameToken>
2172                 <wsse:Username> "Nick" </wsse:Username>
2173             </wsse:UsernameToken>
2174         </fed:SecurityToken>
2175         <fed:ProofToken>...</fed:ProofToken>
2176     </fed:Pseudonym>
2177 </wsrt:Value>
2178 </wsrt:Fragment>
2179 </wsrt:Create>
2180 </S:Body>
2181 </S:Envelope>
```

2182

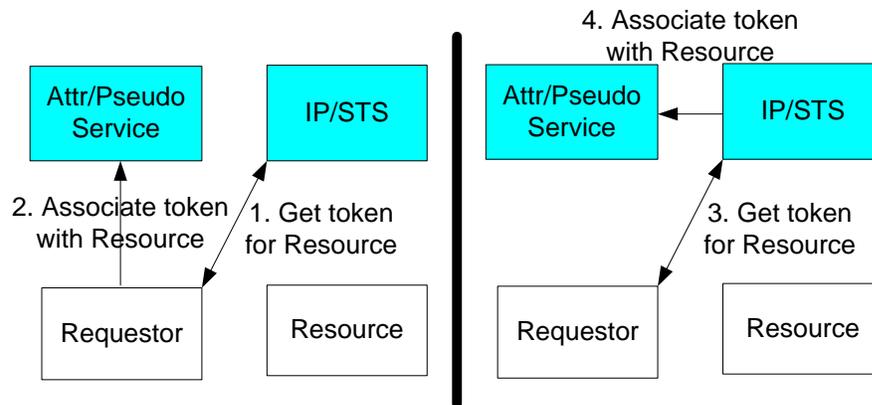
7 Security Tokens and Pseudonyms

2183
2184
2185
2186

As previously mentioned, the pseudonym service MAY 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.

2187
2188
2189
2190
2191
2192
2193

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).



2194

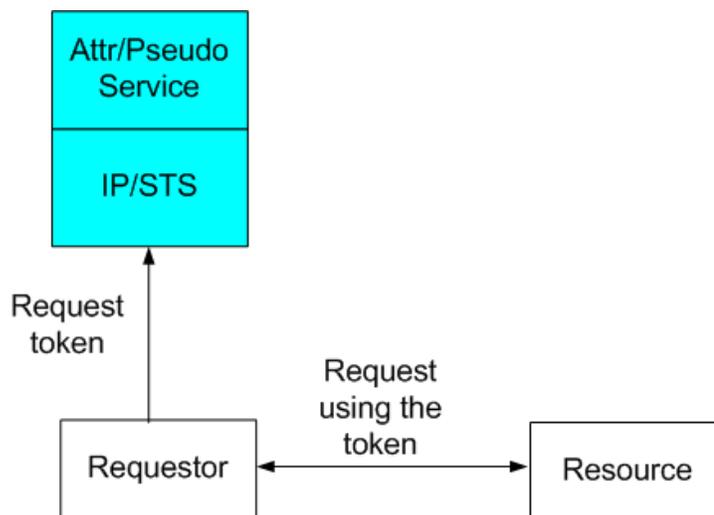
2195

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

2196

2197
2198
2199
2200

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:



2201

2202

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

2203 The pseudonym service SHOULD be self-maintained with respect to valid security tokens. That is,
2204 security tokens that have expired or are otherwise not valid for any reason MAY be automatically
2205 discarded by the service.

2206 This approach is an alternative to having the pseudonym service directly return the security token
2207 issuance. Both approaches SHOULD be supported in order to address different scenarios and
2208 requirements.

2209 The following sub-sections describe how token issuance works for different types of keys.

2210 7.1 RST and RSTR Extensions

2211 With the addition of pseudonyms and the integration of an IP/STS with a pseudonym service, an IP/STS
2212 MAY automatically map pseudonyms based on the target service. If it doesn't, the following additional
2213 options MAY be included in the security token requests using the `<wst:RequestSecurityToken>`
2214 request to explicitly request a mapping or to clarify the type of mapping desired.

2215 The following syntax illustrates the RST extension to support these new options:

```
2216 <fed:RequestPseudonym SingleUse="xs:boolean" ? Lookup="xs:boolean" ? ...>  
2217 ...  
2218 </fed:RequestPseudonym>
```

2219 `/fed:RequestPseudonym`

2220 This OPTIONAL element MAY be specified in a `<wst:RequestSecurityToken>` request to
2221 indicate how pseudonyms are to be processed for the requested token.

2222 `/fed:RequestPseudonym/@SingleUse`

2223 This optional OPTIONAL attribute indicates if a single-use pseudonym is returned (true), or if the
2224 service uses a constant identifier (false – the default).

2225 `/fed:RequestPseudonym/@Lookup`

2226 This OPTIONAL attribute indicates if an associated pseudonym for the specified scope is used
2227 (true – the default) or if the primary identity is used even if an appropriate pseudonym is
2228 associated (false).

2229 `/fed:RequestPseudonym/{any}`

2230 This is an extensibility mechanism to allow additional information to be specified. Use of this
2231 extensibility mechanism MUST NOT alter the semantics defined in this specification.

2232 `/fed:RequestPseudonym/@{any}`

2233 This is an extensibility mechanism to allow additional attributes to be specified. Use of this
2234 extensibility mechanism MUST NOT alter the semantics defined in this specification.

2235 If the `<RequestPseudonym>` isn't present, pseudonym usage/lookup and single use is at the discretion
2236 of the IP/STS. Note that if present, as with all RST parameters, processing is at the discretion of the STS
2237 and it MAY choose to use its own policy instead of honoring the requestor's parameters.

2238 Note that the above MAY be echoed in a RSTR response confirming the value used by the STS.

2239 7.2 Usernames and Passwords

2240 If an IP/STS returns a security token based on a username, then the token can be stored in the
2241 pseudonym service.

2242 If a corresponding password is issued (or if the requestor specified one), then it too MAY be stored with
2243 the pseudonym and security token so that it can be returned as the proof-of-possession token in the
2244 RSTR response.

2245 If a pseudonym is present, but no security token is specified, then the IP/STS MAY return a
2246 <UsernameToken> in the RSTR response to indicate the pseudonym.

2247 **7.3 Public Keys**

2248 Generally, when an IP/STS issues a new security token with public key credentials, the public key in the
2249 new security token is the same as the key in the provided input security token thereby allowing the same
2250 proof (private key) to be used with the new token since the public key is the same. In such cases, the
2251 new token can be saved directly.

2252 If, however, the IP/STS issues a new public key (and corresponding private key), then the private key
2253 MAY be stored with the pseudonym as a proof token so that it can be subsequently returned as the proof-
2254 of-possession token in the RSTR response.

2255 **7.4 Symmetric Keys**

2256 If an IP/STS returns a token based on a symmetric key (and the corresponding proof information), then
2257 the proof information MAY be stored with the pseudonym and token so that it can be used to construct a
2258 proof-of-possession token in the RSTR response.

2259

8 Additional WS-Trust Extensions

2260

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

2261

8.1 Reference Tokens

2262

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.

2263

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.

2264

2265

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.

2266

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

2272

```
http://docs.oasis-open.org/wsfed/federation/200706/reftoken
```

2274

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

2275

```
<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>
```

2276

2277

2278

2279

2280

2281

2282

/fed:ReferenceToken

2283

2284

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

2285

/fed:ReferenceToken/fed:ReferenceEPR

2286

2287

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

2288

/fed:ReferenceToken/fed:ReferenceDigest

2289

2290

2291

2292

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.

2293

/fed:ReferenceToken/fed:ReferenceDigest/@{any}

2294

2295

This extensibility mechanism allows additional attributes to be specified. Use of this extensibility mechanism MUST NOT alter the semantics defined in this specification.

2296

/fed:ReferenceToken/fed:ReferenceType

2297

2298

2299

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.

2300

/fed:ReferenceToken/fed:ReferenceType/@{any}

2301 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2302 mechanism MUST NOT alter the semantics defined in this specification.

2303 /fed:ReferenceToken/fed:SerialNo

2304 An OPTIONAL URI value that uniquely identifies the reference token.

2305 /fed:ReferenceToken/fed:SerialNo/{any}

2306 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2307 mechanism MUST NOT alter the semantics defined in this specification.

2308 /fed:ReferenceToken/{any}

2309 This extensibility mechanism allows additional informative elements to be specified Use of this
2310 extensibility mechanism MUST NOT alter the semantics defined in this specification.

2311 /fed:ReferenceToken/{any}

2312 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2313 mechanism MUST NOT alter the semantics defined in this specification.

2314 There are no requirements on the security associated with the handle or dereferencing it. If the resulting
2315 token is secured or does not contain sensitive information the STS MAY just make it openly accessible.
2316 Alternatively, the STS MAY use the <wsp:AppliesTo> information from the RST to secure the token
2317 such that only requestors that can speak for that address can obtain the token.

2318 8.2 Indicating Federations

2319 In some scenarios an STS, resource provider, or service provider MAY be part of multiple federations and
2320 allow token requests at a single endpoint that could be processed in the context of any of the federations
2321 (so long as the requestor is authorized). In such cases, there may be a need for the requestor to identify
2322 the federation context in which it would like the token request to be processed.

2323 The following <fed:FederationID> element can be included in a RST (as well as an RSTR):

```
2324 <fed:FederationID ...>xs:anyURI</fed:FederationID>
```

2325 /fed:FederationID

2326 This element identifies the federation context as a URI value in which the token request is made
2327 (or was processed).

2328 /fed:FederationID/{any}

2329 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2330 mechanism MUST NOT alter the semantics defined in this specification.

2331 Note that if a FederationID is not specified, the *default* federation is assumed.

2332 8.3 Obtaining Proof Tokens from Validation

2333 A requestor may obtain a token for a federation for which the recipient service doesn't actually have the
2334 rights to use and extract the session key. For example, when a requestor's IP/STS and the recipient's
2335 IP/STS have an arrangement and share keys but the requestor and recipient only describe federation
2336 between themselves. In such cases, the requestor and the recipient MUST obtain the session keys
2337 (proof tokens) from their respective IP/STS. For the requestor this is returned in the proof token of its
2338 request.

2339 For the recipient, it must pass the message to its IP/STS to have it validated. As part of the validation
2340 process, the proof token MAY be requested by including the parameter below in the RST. When this
2341 element is received by an IP/STS, it indicates a desire to have a <wst:RequestedProofToken>
2342 returned with the session key so that the recipient does not have to submit subsequent messages for
2343 validation.

2344 The syntax of the `<fed:RequestProofToken>` is as follows:

```
2345 <fed:RequestProofToken ...>
2346   ...
2347 </fed:RequestProofToken>
```

2348 `/fed:RequestProofToken`

2349 When used with a *Validate* request this indicates that the requestor would like the STS to return a
2350 proof token so that subsequent messages using the same token/key can be processed by the
2351 recipient directly.

2352 `/fed:RequestProofToken/@{any}`

2353 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2354 mechanism MUST NOT alter the semantics defined in this specification.

2355 `/fed:RequestProofToken/{any}`

2356 This contents of this element are undefined and MAY be extended. Use of this extensibility
2357 mechanism MUST NOT alter the semantics defined in this specification.

2358

2359 8.4 Client-Based Pseudonyms

2360 Previous sections have discussed requesting pseudonyms based on registered identities. In some cases
2361 a requestor desires a pseudonym to be issued using *ad hoc* data that is specifies as an extension to the
2362 RST request. As with all WS-Trust parameters, the IP/STS is NOT REQUIRED to honor the parameter,
2363 but if it does, it SHOULD echo the parameter in the RSTR.

2364 A requestor MAY specify the `<fed:ClientPseudonym>` element to indicate pseudonym information it
2365 would like used in the issued token. The STS MUST accept all of the information or none of it. That is, it
2366 MUST NOT use some pseudonym information but not other pseudonym information.

2367 The syntax of the `<fed:ClientPseudonym>` element is as follows:

```
2368 <fed:ClientPseudonym ...>
2369   <fed:PPID ...>xs:string</fed:PPID> ?
2370   <fed:DisplayName ...>xs:string</fed:DisplayName> ?
2371   <fed:Email ...>xs:string</fed:EMail> ?
2372   ...
2373 </fed:ClientPseudonym>
```

2374 `/fed:ClientPseudonym`

2375 This indicates a request to use specific identity information in resulting security tokens.

2376 `/fed:ClientPseudonym/fed:PPID`

2377 If the resulting security token contains any form of private personal identifier, this string value is to
2378 be used as the basis. The issuer MAY use this value as the input (a seed) to a custom function
2379 and the result used in the issued token.

2380 `/fed:ClientPseudonym/fed:PPID/@{any}`

2381 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2382 mechanism MUST NOT alter the semantics defined in this specification.

2383 `/fed:ClientPseudonym/fed:DisplayName`

2384 If the resulting security token contains any form of display or subject name, this string value is to
2385 be used.

2386 `/fed:ClientPseudonym/fed:DisplayName/@{any}`

2387 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2388 mechanism MUST NOT alter the semantics defined in this specification.

2389 /fed:ClientPseudonym/fed:EMail

2390 If the resulting security token contains any form electronic mail address, this string value is to be
2391 used.

2392 /fed:ClientPseudonym/fed:EMail/{any}

2393 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2394 mechanism MUST NOT alter the semantics defined in this specification.

2395 /fed:ClientPseudonym/{any}

2396 This extensibility point allows other pseudonym information to be specified. If the STS does not
2397 understand any element it MUST either ignore the entire <fed:ClientPseudonym> or Fault.

2398 /fed:ClientPseudonym/@{any}

2399 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2400 mechanism MUST NOT alter the semantics defined in this specification.

2401 8.5 Indicating Freshness Requirements

2402 There are times when a token requestor desires to limit the age of the credentials used to authenticate.
2403 The parameter MAY be specified in a RST to indicate the desired upper bound on credential age. As well
2404 this parameter is used to indicate if the requestor is willing to allow issuance based on cached
2405 credentials.

2406 The syntax of the <fed:Freshness> element is as follow:

```
2407 <fed:Freshness AllowCache="xs:boolean" ...>  
2408   xs:unsignedInt  
2409 </fed:Freshness>
```

2410 /fed:Freshness

2411 This indicates a desire to limit the age of authentication credentials. This REQUIRED unsigned
2412 integer value indicates the upper bound on credential age specified in minutes only. A value of
2413 zero (0) indicates that the STS is to immediately verify identity if possible or use the minimum age
2414 credentials possible if immediate (e.g. interactive) verification is not possible. If the `AllowCache`
2415 attribute is specified, then the cached credentials SHOULD meet the freshness time window.

2416 /fed:Freshness/@{any}

2417 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2418 mechanism MUST NOT alter the semantics defined in this specification.

2419 /fed:Freshness/@AllowCache

2420 This OPTIONAL Boolean qualifier indicates if cached credentials are allowed. The default value
2421 is *true* indicating that cached information MAY be used. If *false* the STS SHOULD NOT use
2422 cached credentials in processing the request.

2423 If the credentials provided are valid but do not meet the freshness requirements, then the
2424 `fed:NeedFresherCredentials` fault MUST be returned informing the requestor that they need to
2425 obtain fresher credentials in order to process their request.

2426 9 Authorization

2427 An authorization service is a specific instance of a security token service (STS). To ensure consistent
2428 processing and interoperability, this specification defines a common model for authorization services, a
2429 set of extensions enabling rich authorization, and a common profile of [WS-Trust] to facilitate
2430 interoperability with authorization services.

2431 This section describes a model and two extensions specific to rich authorization. The first allows
2432 additional context information to be provided in authorization requests. The second allows services to
2433 indicate that additional claims are required to successfully process specific requests.

2434 9.1 Authorization Model

2435 An authorization service is an STS that operates in a decision brokering process. That is, it receives a
2436 request (either directly or on behalf of another party) for a token (or set of tokens) to access another
2437 service. Such a service MAY be separate from the target service or it MAY be co-located. The
2438 authorization service determines if the requested party can access the indicated service and, if it can,
2439 issues a token (or set of tokens) with the allowed rights at the specified service. These two aspects are
2440 distinct and could be performed by different collaborating services.

2441 In order to make the authorization decision, the authorization service MUST ensure that the requestor has
2442 presented and proven the claims required to access the target service (or resource) indicated in the
2443 request (e.g. in the `<wsp:AppliesTo>` parameter). Logically, the authorization service constructs a
2444 table of name/value pairs representing the claims required by the target service. The logical *requirement*
2445 *table* is constructed from the following sources and may MAY be supplemented by additional service
2446 resources:

- 2447 • The address of the EPR for the target service
- 2448 • The reference properties from the EPR of the target service
- 2449 • Parameters of the RST
- 2450 • External access control policies

2451 Similarly, the claim table is a logical table representing the claims and information available for the
2452 requestor that the authorization service uses as the basis for its decisions. This logical table is
2453 constructed from the following sources:

- 2454 • Proven claims that are bound to the RST request (both primary and supporting)
- 2455 • Supplemental authorization context information provided in the request
- 2456 • External authorization policies

2457 9.2 Indicating Authorization Context

2458 In the [WS-Trust] protocol, the requestor of a token conveys the desired properties of the required token
2459 (such as the token type, key type, claims needed, etc.) in the token request represented by the RST
2460 element. Each such property is represented by a child element of the RST, and is typically specified by
2461 the Relying Party that will consume the issued token in its security policy assertion as defined by [WS-
2462 SecurityPolicy]. The token properties specified in a token request (RST) generally translate into some
2463 aspect of the content of the token that is issued by a STS. However, in many scenarios, there is a need to
2464 be able to convey additional contextual data in the token request that influences the processing and token
2465 issuance behavior at the STS. The supplied data MAY (but need not) directly translate into some aspect
2466 of the actual token content.

2467 To enable this a new element, `<auth:AdditionalContext>`, is defined to provide additional context
2468 information. This MAY be specified in RST requests and MAY be included in RSTR responses.

2469 The syntax is as follows:

```
2470 <wst:RequestSecurityToken>  
2471 ...  
2472 <auth:AdditionalContext>  
2473   <auth:ContextItem Name="xs:anyURI" Scope="xs:anyURI" ? ...>  
2474     (<auth:Value>xs:string</auth:Value> |  
2475     xs:any ) ?  
2476   </auth:ContextItem> *  
2477   ...  
2478 </auth:AdditionalContext>  
2479 ...  
2480 </wst:RequestSecurityToken>
```

2481 The following describes the above syntax:

2482 `/auth:AdditionalContext`

2483 This OPTIONAL element provides additional context for the authorization decision (which
2484 determines token issuance).

2485 `/auth:AdditionalContext/ContextItem`

2486 This element is provides additional authorization context as simple name/value pairs. Note that
2487 this is the only `fed:AdditionalContext` element defined in this specification.

2488 `/auth:AdditionalContext/ContextItem/@Name`

2489 This REQUIRED URI attribute specifies the kind of the context item being provided. There are no
2490 pre-defined context names.

2491 `/auth:AdditionalContext/ContextItem/@Scope`

2492 This OPTIONAL URI attribute specifies the scope of the context item. That is, the subject of the
2493 context item. If this is not specified, then the scope is undefined.

2494 The following scopes a pre-defined but others MAY be added:

URI	Description
<code>http://docs.oasis-open.org/wsfed/authorization/200706/ctx/requestor</code>	The context item applies to the requestor of the token (or the <code>OnBehalfOf</code>)
<code>http://docs.oasis-open.org/wsfed/authorization/200706/ctx/target</code>	The context item applies to the intended target (<code>AppliesTo</code>) of the token
<code>http://docs.oasis-open.org/wsfed/authorization/200706/ctx/action</code>	The context item applies to the intended action at the intended target (<code>AppliesTo</code>) of the token

2495 `/auth:AdditionalContext/ContextItem/Value`

2496 This OPTIONAL string element specifies the simple string value of the context item.

2497 `/auth:AdditionalContext/ContextItem/{any}`

2498 This OPTIONAL element allows a custom context value to be associated with the context item.
2499 This MUST NOT be specified along with the `Value` element (there can only be a single value).

2500 /auth:AdditionalContext/ContextItem/@{any}

2501 This extensibility point allows additional attributes to be specified. Use of this extensibility
2502 mechanism MUST NOT violate any semantics defined in this document.

2503 /auth:AdditionalContext/@{any}

2504 This extensibility point allows additional attributes. Use of this extensibility mechanism MUST
2505 NOT violate any semantics defined in this document.

2506 /auth:AdditionalContext/{any}

2507 This element has an open content model allowing different types of context to be specified. That
2508 is, custom elements can be defined and included so long as all involved parties understand the
2509 elements.

2510 An example of an RST token request where this element is used to specify additional context data is
2511 given below. Note that this example specifies claims using a custom dialect.

```
2512 <wst:RequestSecurityToken>  
2513   <wst:TokenType>  
2514     urn:oasis:names:tc:SAML:1.0:assertion  
2515   </wst:TokenType>  
2516   <wst:RequestType>  
2517     http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue  
2518   </wst:RequestType>  
2519   <wst:Claims Dialect="...">  
2520     ...  
2521   </wst:Claims>  
2522   ...  
2523   <auth:AdditionalContext>  
2524     <auth:ContextItem Name="urn:...:PurchaseAmount">  
2525       <auth:Value>125.00</auth:Value>  
2526     </auth:ContextItem>  
2527     <auth:ContextItem Name="urn:...:MerchantId">  
2528       <auth:Value>FABRIKAM 92305645883256</auth:Value>  
2529     </auth:ContextItem>  
2530   </auth:AdditionalContext>  
2531 </wst:RequestSecurityToken>
```

2532 9.3 Common Claim Dialect

2533 There are different claim representations that are used across different Web Service implementations
2534 making it difficult to express claims in a common interoperable way. To facilitate interoperability, this
2535 section defines a simple dialect for expressing claims in a format-neutral way. This new dialect uses the
2536 <auth:ClaimType> element for representing a claim, and the dialect is identified by the following URI:

```
2537 http://docs.oasis-open.org/wsfed/authorization/200706/authclaims
```

2538 This dialect MAY be used within the <wst:Claims> element when making token requests or in
2539 responses. This dialect MAY also be used in describing a service's security requirements using [WS-
2540 SecurityPolicy]. Note that the xml:lang attribute MAY be used where allowed via attribute extensibility to
2541 specify a language of localized elements and attributes using the language codes specified in [RFC
2542 3066].

2543 The syntax for the <auth:ClaimType> element for representing a claim is as follows:

```
2544 <auth:ClaimType Uri="xs:anyURI" Optional="xs:boolean">  
2545   <auth:DisplayName ...> xs:string </auth:DisplayName> ?  
2546   <auth:Description ...> xs:string </auth:Description> ?  
2547   <auth:DisplayValue ...> xs:string </auth:DisplayValue> ?  
2548   (<auth:Value>...</auth:Value> |  
2549   <auth:StructuredValue ...>...</auth:StructuredValue> |
```

2550
2551
2552
2553
2554
2555

```
<auth:EncryptedValue @DecryptionCondition="xs:anyURI">
  <xenc:EncryptedData>...</xenc:EncryptedData>
</auth:EncryptedValue> |
<auth:ConstrainedValue>...</auth:ConstrainedValue> ?
...
</auth:ClaimType>
```

2556 The following describes the above syntax:

2557 /auth:ClaimType

2558 This element represents a specific claim.

2559 /auth:ClaimType/@Uri

2560 This REQUIRED URI attribute specifies the kind of the claim being indicated. The following claim
2561 type is pre-defined, but other types MAY be defined:

URI	Description
http://docs.oasis-open.org/wsfed/authorization/200706/claims/action	The wsa:Action specified in a request

2562 /auth:ClaimType/@Optional

2563 This OPTIONAL boolean attribute specifies the claim is optional (true) or required (false). The
2564 default value is false.

2565 /auth:ClaimType/auth:DisplayName

2566 This OPTIONAL element provides a friendly name for this claim type that can be shown in user
2567 interfaces.

2568 /auth:ClaimType/auth:DisplayName/@{any}

2569 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
2570 NOT alter the semantics defined in this specification.

2571 /auth:ClaimType/auth:Description

2572 This OPTIONAL element provides a description of the semantics for this claim type.

2573 /auth:ClaimType/auth:Description/@{any}

2574 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
2575 NOT alter the semantics defined in this specification.

2576 /auth:ClaimType/auth:DisplayValue

2577 This OPTIONAL element provides a displayable value for a claim returned in a security token.

2578 /auth:ClaimType/auth:DisplayValue/@{any}

2579 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
2580 NOT alter the semantics defined in this specification.

2581 /auth:ClaimType/auth:Value

2582 This OPTIONAL element allows a specific string value to be specified for the claim.

2583 /auth:ClaimType/auth:EncryptedValue

2584 This OPTIONAL element is used to convey the ciphertext of a claim.

2585 /auth:Claims/auth:ClaimType/auth:EncryptedValue/xenc:EncryptedData

2586 This OPTIONAL element is only used for conveying the KeyInfo.

2587 /auth:Claims/auth:ClaimType/auth:EncryptedValue/@DecryptionCondition
 2588 This OPTIONAL attribute specifies the URI indicating the conditions under which this claim
 2589 SHOULD be decrypted.
 2590 The decryptor SHOULD decrypt only if the decryption condition is fulfilled. Note that a decryptor
 2591 MAY be a 3rd party. In order for such a decryption to happen, the recipient of the claim has to
 2592 provide the ciphertext and decryption condition to the decryptor.. This specification does not
 2593 define any URI values. Participating parties MAY use other values under private agreements.

2594 /auth:ClaimType/auth:StructuredValue
 2595 This OPTIONAL element specifies the value of a claim in a well formed xml structure.

2596 /auth:ClaimType/auth:StructuredValue/@{any}
 2597 This extensibility point allows additional structured value types to be specified for the claim. Use
 2598 of this extensibility point MUST NOT alter the semantics defined in this specification.

2599

2600 /auth:ClaimType/auth:ConstrainedValue
 2601 This OPTIONAL element specifies constraints on a given claim. It MAY contain the constraint that
 2602 value MUST satisfy, or it MAY contain the actual constrained value. For more details on
 2603 constraints see section 9.3.1.

2604 /auth:ClaimType/@{any}
 2605 This extensibility point allows attributes to be added. Use of this extensibility point MUST NOT
 2606 alter the semantics defined in this specification.

2607 /auth:ClaimType/{any}
 2608 This extensibility point allows additional values types to be specified for the claim. Use of this
 2609 extensibility point MUST NOT alter the semantics defined in this specification.

2610

2611 9.3.1 Expressing value constraints on claims

2612 When requesting or returning claims in a [WS-Trust] RST request or specifying required claims in [WS-
 2613 SecurityPolicy] it MAY be necessary to express specific constraints on those claims. The
 2614 <auth:ConstrainedValue> element, used within the <auth:ClaimType> element, provides this
 2615 capability.

2616

2617 The semantics of the comparison operators specified in the <auth:ConstrainedValue> element are
 2618 specific to the given claim type unless explicitly defined below.

2619

2620 The syntax for the <auth:ConstrainedValue> element, used within the <auth:ClaimType>
 2621 element, is as follows.

```

2622 <auth:ConstrainedValue AssertConstraint="xs:boolean">
2623   ( <auth:ValueLessThan>
2624     (<auth:Value> xs:string </auth:Value> |
2625     <auth:StructuredValue> xs:any </auth:StructuredValue>)
2626   </auth:ValueLessThan> |
2627   <auth:ValueLessThanOrEqual>
2628     (<auth:Value> xs:string </auth:Value> |
2629     <auth:StructuredValue> xs:any </auth:StructuredValue>)
2630   </auth:ValueLessThanOrEqual> |
2631   <auth:ValueGreaterThan>
2632     (<auth:Value> xs:string </auth:Value> |
2633     <auth:StructuredValue> xs:any </auth:StructuredValue>)
  
```

```

2634     </auth:ValueGreaterThan> |
2635     <auth:ValueGreaterThanOrEqual>
2636         (<auth:Value> xs:string </auth:Value> |
2637         <auth:StructuredValue> xs:any </auth:StructuredValue>)
2638     </auth:ValueGreaterThanOrEqual> |
2639     <auth:ValueInRange>
2640         <auth:ValueUpperBound>
2641             (<auth:Value> xs:string </auth:Value> |
2642             <auth:StructuredValue> xs:any </auth:StructuredValue>)
2643         </auth:ValueUpperBound>
2644         <auth:ValueLowerBound>
2645             (<auth:Value> xs:string </auth:Value> |
2646             <auth:StructuredValue> xs:any </auth:StructuredValue>)
2647         </auth:ValueLowerBound>
2648     </auth:ValueInRange> |
2649     <auth:ValueOneOf>
2650         (<auth:Value> xs:string </auth:Value> |
2651         <auth:StructuredValue> xs:any </auth:StructuredValue>) +
2652     </auth:ValueOneOf> ) ?
2653     ...
2654 </auth:ConstrainedValue> ?

```

2655 The following describe the above syntax

2656 /auth:ClaimType/auth:ConstrainedValue

2657 This OPTIONAL element indicates that there are constraints on the claim value. This element
2658 MUST contain one of the defined elements below when used in a RST/RSTR message. This
2659 element MAY be empty when used in the fed:ClaimTypesOffered element to describe a service's
2660 capabilities which means that any constrained value form, from he defined elements below, is
2661 supported for the claim type.

2662 /auth:ClaimType/auth:ConstrainedValue/@AssertConstraint

2663 This OPTIONAL attribute indicates that when a claim is issued the constraint itself is asserted
2664 (when true) or that a value that adheres to the condition is asserted (when false). The default
2665 value is true.

2666 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThan

2667 This OPTIONAL element indicates that the value of the claim MUST be less than the given value.

2668 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThan/auth:Value

2669 This element specifies the string value the claim MUST be less than.

2670 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThan/auth:StructuredValue

2671 This element specifies the value of a claim in a well formed xml structure the claim MUST be less
2672 than.

2673 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThanOrEqual

2674 This OPTIONAL element indicates that the value of the claim MUST be less than or equal to the
2675 given value.

2676 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThanOrEqua/auth:Value

2677 This element specifies the string value the claim MUST be less than or equal to.

2678 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThanOrEqual/auth:StructuredValue

2679 This element specifies the value of a claim in a well formed xml structure the claim MUST be less
2680 than or equal to.

2681 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThan

2682 This OPTIONAL element indicates that the value of the claim MUST be greater than the given
2683 value.

2684 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThan/auth:Value
2685 This element specifies the string value the claim MUST be greater than.

2686 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThan/auth:StructuredValue
2687 This element specifies the value of a claim in a well formed xml structure the claim MUST be
2688 greater than.

2689 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThanOrEqual
2690 This OPTIONAL element indicates that the value of the claim MUST be greater than or equal to
2691 the given value.

2692 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThanOrEqual/auth:Value
2693 This element specifies the string value the claim MUST be greater than or equal to.

2694 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThanOrEqual/auth:StructuredValue
2695 This element specifies the value of a claim in a well formed xml structure the claim MUST be
2696 greater than or equal to.

2697 /auth:ClaimType/auth:ConstrainedValue/auth:ValueInRange
2698 This OPTIONAL element indicates that the value of the claim MUST be in the specified range.
2699 The specified boundary values are included in the range.

2700 /auth:ClaimType/auth:ConstrainedValue/auth:ValueInRange/auth:ValueUpperBound
2701 This element specifies the upper limit on a given value.

2702 /auth:ClaimType/auth:ConstrainedValue/auth:ValueInRange/auth:ValueLowerBound
2703 This element specifies the lower limit on a given value.

2704 /auth:ClaimType/auth:ConstrainedValue/auth:ValueOneOf
2705 This element specifies a collection of values among which the value of claim MUST fall.

2706 /auth:ClaimType/auth:ConstrainedValue/auth:ValueOneOf/auth:Value
2707 This element specifies an allowed string value for the claim.

2708 /auth:ClaimType/auth:ConstrainedValue/auth:ValueOneOf/auth:StructuredValue
2709 This element specifies an allowed value for the claim in a well formed xml structure.

2710 /auth:ClaimType/auth:ConstrainedValue/{any}
2711 This extensibility point allows additional constrained value types to be specified for the claim..
2712 Use of this extensibility mechanism MUST NOT alter the semantics defined in this specification.

2713
2714

2715 9.4 Claims Target

2716 The @fed:ClaimsTarget attribute is defined for use on the wst:Claims element as a way to indicate the
2717 intended consumer of claim information .

2718 The syntax for @auth:ClaimsTarget is as follows.

```
2719 <wst:Claims fed:ClaimsTarget="..." ...>
2720 ...
2721 </wst:Claims>
```

2722 The following describes the above syntax.

2723

2724 /wst:Claims /@fed:ClaimsTarget

2725 This OPTIONAL attribute indicates the intended consumer of the claim information. If this
2726 attribute is not specified, then a default value is assumed. The predefined values are listed in the
2727 table below, but parties MAY use other values under private agreements. This attribute MAY be
2728 used if the context doesn't provide a default target or if a different target is required. This attribute
2729 MUST NOT appear in a RST or RSTR message defined in WS-Trust,

2730

URI	Description
<code>http://docs.oasis-open.org/wsfed/authorization/200706/claims/target/recipient</code> (default)	Whoever is the ultimate receiver of the element is expected to process it.
<code>http://docs.oasis-open.org/wsfed/authorization/200706/claims/target/client</code>	The client or originating requestor (typically the party issuing the original RST request) is expected to process this element.
<code>http://docs.oasis-open.org/wsfed/authorization/200706/claims/target/issuer</code>	The entity that has the responsibility and (typically the party issuing the token) is expected to process this element.
<code>http://docs.oasis-open.org/wsfed/authorization/200706/claims/target/rp</code>	The entity that is expected to consume a security token is expected to process this element.

2731

2732

2733 9.5 Authorization Requirements

2734 Authorization requestors and issuing services (providers) compliant with this specification MUST conform
2735 to the rules described in this section when issuing RST requests and returning RSTR responses.

2736 *R001* – The authorization service MUST accept an `<wsp:AppliesTo>` target in the RST

2737 *R002* – The authorization service MUST specify an `<wsp:AppliesTo>` target in the RSTR if one is
2738 specified in the RST

2739 *R003* – The authorization service SHOULD encode the `<wsp:AppliesTo>` target in issued tokens if the
2740 token format supports it

- 2741 *R004* – The `<wsp:AppliesTo>` target for issued token MAY be for a broader scope than the scope
2742 specified in the RST but MUST NOT be narrower (as specified in WS-Trust)
- 2743 *R005* – The authorization service MUST accept reference properties in the `<wsp:AppliesTo>` target
- 2744 *R006* – The authorization service MUST accept the `<auth:AdditionalContext>` parameter
- 2745 *R007* – The authorization service MUST accept the claim dialect defined in this specification
- 2746 *R008* – The authorization service MAY ignore elements in the `auth:AdditionalContext` parameter if it
2747 doesn't recognize or understand them

2748

10 Indicating Specific Policy/Metadata

2749 When a requestor communicates with a recipient service there may be additional security requirements,
2750 beyond those in the general security policy or other metadata, that are required based on the specifics of
2751 the request. For example, if a request contains a “gold customer” custom message header to indicate
2752 customer classification (and routing), then proof that the requestor is a gold member may be required
2753 when the request is actually authorized. There may also be contextual requirements which are hard to
2754 express in a general policy. For example, if a requestor wants to submit a purchase, it may be required to
2755 present a token from a trusted source attesting that the requestor has the requisite funds.

2756 To address this scenario a mechanism is introduced whereby the recipient service MAY indicate to the
2757 requestor that additional security semantics apply to the request. The requestor MAY reconstruct the
2758 message in accordance with the new requirements if it can do so. In some cases the requestor may
2759 need to obtain additional tokens from an authorization or identity service and then reconstruct and
2760 resubmit the message.

2761 The mechanism defined by this specification that MAY be used to dynamically indicate that a specific
2762 policy or metadata applies to a specific request is to issue a specialized SOAP Fault. This fault indicates
2763 to the requestor that additional security metadata is REQUIRED. The new metadata, in its complete form
2764 (not a delta) is specified in the fault message using the WS-MetadataExchange format.

2765 The fault is the `fed:SpecificMetadata` and is specified as the fault code. The `<S:Detail>` of this
2766 fault contains a `mex:Metadata` element containing sections with the effective metadata for the endpoint
2767 processing this specific request.

2768 The following example illustrates a fault with embedded policy:

```
2769 <S:Envelope xmlns:S="..." xmlns:auth="..." xmlns:wst="..." xmlns:fed="..."  
2770   xmlns:sp="..." xmlns:wsp="..." xmlns:mex="...">  
2771   <S:Body>  
2772     <S:Fault>  
2773       <S:Code>  
2774         <S:Value>fed:SpecificMetadata</S:Value>  
2775       </S:Code>  
2776       <S:Reason>  
2777         <S:Text>Additional credentials required in order to  
2778           perform operation. Please resubmit request with  
2779           appropriate credentials.  
2780       </S:Text>  
2781     </S:Reason>  
2782     <S:Detail>  
2783       <mex:Metadata>  
2784         <mex:MetadataSection  
2785           Dialect='http://schemas.xmlsoap.org/ws/2004/09/policy'>  
2786         <wsp:Policy>  
2787           ...  
2788         <sp:EndorsingSupportingTokens>  
2789           <sp:IssuedToken>  
2790             <sp:Issuer>...</sp:Issuer>  
2791             <sp:RequestSecurityTokenTemplate>  
2792               <wst:Claims>  
2793                 ...  
2794               </wst:Claims>  
2795             <auth:AdditionalContext>  
2796               ...  
2797             </auth:AdditionalContext>  
2798           ...
```

```
2799         </sp:RequestSecurityTokenTemplate>
2800     </sp:IssuedToken>
2801     </sp:EndorsingSupportingTokens>
2802     </wsp:Policy>
2803     </mex:MetadataSection>
2804     </mex:Metadata>
2805     </S:Detail>
2806     </S:Fault>
2807     </S:Body>
2808 </S:Envelope>
```

2809

11 Authentication Types

2810 The [WS-Trust] specification defines the `wst:AuthenticationType` parameter to indicate a desired
2811 type of authentication (or to return the type of authentication verified). However, no pre-defined values
2812 are specified. While any URI can be used, to facilitate federations the following OPTIONAL types are
2813 defined but are NOT REQUIRED to be used:

URI	Description
<code>http://docs.oasis-open.org/wsfed/authorization/200706/authntypes/unknown</code>	Unknown level of authentication
<code>http://docs.oasis-open.org/wsfed/authorization/200706/authntypes/default</code>	Default sign-in mechanisms
<code>http://docs.oasis-open.org/wsfed/authorization/200706/authntypes/Ssl</code>	Sign-in using SSL
<code>http://docs.oasis-open.org/wsfed/authorization/200706/authntypes/SslAndKey</code>	Sign-in using SSL and a security key
<code>http://docs.oasis-open.org/wsfed/authorization/200706/authntypes/SslAndStrongPassword</code>	Sign-in using SSL and a “strong” password
<code>http://docs.oasis-open.org/wsfed/authorization/200706/authntypes/SslAndStrongPasswordWithExpiration</code>	Sign-in using SSL and a “strong” password with expiration
<code>http://docs.oasis-open.org/wsfed/authorization/200706/authntypes/smartcard</code>	Sign-in using Smart Card

2814

2815

12 Privacy

2816 When a requestor contacts an authority to obtain a security token or to obtain authorization for an action it
2817 is often the case that information subject to personal or organizational privacy requirements MAY be
2818 presented in order to authorize the request. In such cases the authority MAY require the requestor to
2819 indicate the restrictions it expects on the use and distribution of sensitive information contained in tokens
2820 it obtains. In this document, this is referred to as a “disclosure constraint”. It should be noted that
2821 disclosure constraints may apply if the requestor is requesting tokens for itself or if the requestor is acting
2822 on behalf of another party.

2823 This specification describes how requestors can communicate their disclosure constraints to security
2824 token services using the [WS-Trust] protocol. It additionally facilitates the requestor’s compliance with
2825 such constraints by allowing it to request elevated data protection for some or all of the response and
2826 issued tokens. The disclosure constraint and protection elevation request are communicated using
2827 existing WS-Trust mechanisms as well as extensions defined in this specification.

2828 The WS-Trust specification describes how to request tokens as well as parameters to the token request
2829 (RST) for indicating how to encrypt proof information as well as algorithms to be used. The following sub-
2830 sections define extension parameters that MAY be specified in RST requests (and echoed in RSTR
2831 responses) to indicate additional privacy options which complement the existing WS-Trust parameters.

2832 12.1 Confidential Tokens

2833 The information contained within an issued token MAY be confidential or sensitive. Consequently, the
2834 requestor may wish to have this information protected (confidential) so that only the intended recipient of
2835 the resulting token (or tokens) can access the information.

2836 The [WS-Trust] specification describes how to indicate a key to use if any data in the token is to be
2837 encrypted, but doesn’t specify any mandates around when or what data is to be protected. This
2838 parameter indicates a protection requirement from the requestor (the STS MAY choose to protect data
2839 even if the requestor doesn’t mandate it).

2840 Any protected (encrypted) information is secured using the token specified in the `<wst:Encryption>`
2841 parameter or using a token the recipient knows to be correct for the request.

2842 The following parameters MAY be specified in an RST request (and echoed in an RSTR response) to
2843 indicate that potentially sensitive information in the token be protected:

```
2844 <wst:RequestSecurityToken>  
2845 ...  
2846 <priv:ProtectData ...>  
2847   <wst:Claims ...>...</wst:Claims> ?  
2848   ...  
2849 </priv:ProtectData>  
2850 ...  
2851 </wst:RequestSecurityToken>
```

2852 The following describes the above syntax:

2853 /priv:ProtectData

2854 This OPTIONAL parameter indicates that sensitive information in any resultant tokens MUST be
2855 protected (encrypted). If specific claims are identified they MUST be protected. The issuer MAY
2856 have an out-of-band agreement with the requestor as to what MUST be protected. If not, and if
2857 specific claims are not identified, the issuer MUST protect all claims. The issuer MAY choose to
2858 protect more than just the requested claims.

2859 /priv:ProtectData/@{any}

2860 This extensibility point allows additional attributes to be specified. Use of this extensibility
2861 mechanism MUST NOT violate any semantics defined in this document.

2862 /priv:ProtectData/wst:Claims

2863 This OPTIONAL element allows the requestor to indicate specific claims which, at a minimum,
2864 MUST be protected. This re-uses the claim specification mechanism from [WS-Trust]. Claims
2865 specified in this set MUST be protected. There is no requirement that all claims specified are in
2866 the issued token. That is, claims identified but not issued MAY be ignored by the STS.

2867 /priv:ProtectData/{any}

2868 This extensibility point allows additional content to be specified Use of this extensibility point
2869 MUST NOT violate any semantics defined in this document.

2870 12.2 Parameter Confirmation

2871 The RST request MAY contain a number of parameters indicating a requestor's disclosure constraints
2872 and data protection preferences. The STS MAY choose , (but is is not required) to honor these
2873 preferences and MAY, (or might not) include selected parameters in any RSTR response.

2874 For privacy reasons a requestor may wish to (a) know if privacy preferences (or any RST parameter)
2875 were accepted or not, (b) what default parameter values were used, (c) require that privacy preferences
2876 (or any RST parameter) be honored, and (d) know what the STS is reporting in a token if it is protected
2877 and unreadable by the requestor.

2878 The following parameters MAY be specified in a RST request (and echoed in an RSTR response) to
2879 indicate to support these requirements:

```
2880 <wst:RequestSecurityToken>  
2881 ...  
2882 <priv:EnumerateParameters ...>  
2883 <xs:list itemType='xs:QName' />  
2884 </priv:EnumerateParameters>  
2885 <priv:FaultOnUnacceptedRstParameters ...>  
2886 ...  
2887 </priv:FaultOnUnacceptedRstParameters>  
2888 <priv:EnumerateAllClaims ...>  
2889 ...  
2890 <priv:EnumerateAllClaims ...>  
2891 ...  
2892 </wst:RequestSecurityToken>
```

2893 The following describes the above syntax:

2894 /priv:EnumerateParameters

2895 A RST request MAY include parameters but the STS is not required to honor them. As such
2896 there is no way for the requestor to know what values where used by the STS. This OPTIONAL
2897 parameter provides a way to request the STS to return the values it used for parameters (or Fault
2898 if it refuses) – either taken from the RST or defaulted using internal policy or settings. The
2899 contents of this parameter indicate a list of QNames that represents RST parameters which
2900 MUST be included in the RSTR. That is, each QName listed MUST be present in the RSTR
2901 returned by the STS indicating the value the STS used for the parameter.

2902 /priv:EnumerateParameters/@{any}

2903 This extensibility point allows additional attributes to be specified. Use of this extensibility point
2904 MUST NOT violate any semantics defined in this document.

2905 /priv:FaultOnUnacceptedRstParameters

2906 This OPTIONAL parameter indicates that if any parameters specified in the RST are not accepted
2907 by the STS, then the STS MUST Fault the request (see the Error Code section for the applicable
2908 Fault code). This means that any unknown parameter causes the request to fail. Note that this
2909 includes extension parameters to the RST.

2910 /priv:FaultOnUnacceptedRstParameters/{any}

2911 This extensibility point allows additional attributes to be specified. Use of this extensibility point
2912 MUST NOT violate any semantics defined in this document.

2913 /priv:FaultOnUnacceptedRstParameters/{any}

2914 This extensibility point allows additional content to be specified. Use of this extensibility point
2915 MUST NOT violate any semantics defined in this document.

2916 /priv:EnumerateAllClaims

2917 This OPTIONAL parameter indicates that all claims issued in resulting tokens MUST be identified
2918 in the RSTR so that the requestor can inspect them. The claims are returned in a
2919 <wst:Claims> element in the RSTR.

2920 /priv:EnumerateAllClaims/{any}

2921 This extensibility point allows additional attributes to be specified. Use of this extensibility point
2922 MUST NOT violate any semantics defined in this document.

2923 /priv:EnumerateAllClaims/{any}

2924 This extensibility point allows additional content to be specified. Use of this extensibility point
2925 MUST NOT violate any semantics defined in this document.

2926 12.3 Privacy Statements

2927 Some services offer privacy statements. This specification defines a mechanism by which privacy
2928 statements, in any form of representation, can be obtained using the mechanisms defined in [WS-
2929 Transfer/WS-ResourceTransfer].

2930 The following URI is defined which can be used as a metadata section dialect in [WS-Transfer/WS-
2931 ResourceTransfer]:

```
2932 http://docs.oasis-open.org/wsfed/privacy/200706/privacypolicy
```

2933 As well, the following element can be used to indicate the EPR to which a [WS-Transfer/WS-
2934 ResourceTransfer] GET message can be sent to obtain the privacy policy:

```
2935 <priv:PrivacyPolicyEndpoint SupportsMex="xs:boolean" ?>  
2936   ...endpoint reference value...  
2937 </priv:PrivacyPolicyEndpoint
```

2938 This element is an endpoint-reference as described in [WS-Addressing]. A [WS-Transfer/WS-
2939 ResourceTransfer] GET message can be sent to it to obtain the previously defined privacy policy dialect.
2940 If the SupportsMex attribute is true (the default is false), then a [WS-MetadataExchange] request can be
2941 directed at the endpoint.

2942 Note that no specific privacy policy form is mandated so requestors must inspect the contents of the
2943 returned privacy policy (or policies) to determine if they can process it (them). The privacy policy could be
2944 a complete privacy policy document, a privacy policy document that references other privacy policies, or
2945 even a compact form of a privacy policy. The form of these documents is outside the scope of this
2946 document.

2947 Alternatively, HTTP GET targets can be specified by including a URL with the following federated
2948 metadata statement:

2949
2950

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

2951

13 Web (Passive) Requestors

2952 This specification defines a model and set of messages for brokering trust and federation of identity and
2953 authentication information across different trust realms and protocols. This section describes how this
2954 Federations model is applied to Web requestors such as Web browsers that cannot directly make Web
2955 Service requests.

2956 13.1 Approach

2957 The federation model previously described builds on the foundation established by [WS-Security] and
2958 [WS-Trust]. Typical Web client requestors cannot perform the message security and token request
2959 operations defined in these specifications. Consequently, this section describes the mechanisms for
2960 requesting, exchanging, and issuing security tokens within the context of a Web requestor.

2961 Web requestors use different but philosophically compatible message exchanges. For example, the
2962 resource might act as its own Security Token Service (STS) and not use a separate service (or even URI)
2963 thereby eliminating some steps. It is expected that subsequent profiles can be defined to extend the Web
2964 mechanisms to include additional exchange patterns.

2965 13.1.1 Sign-On

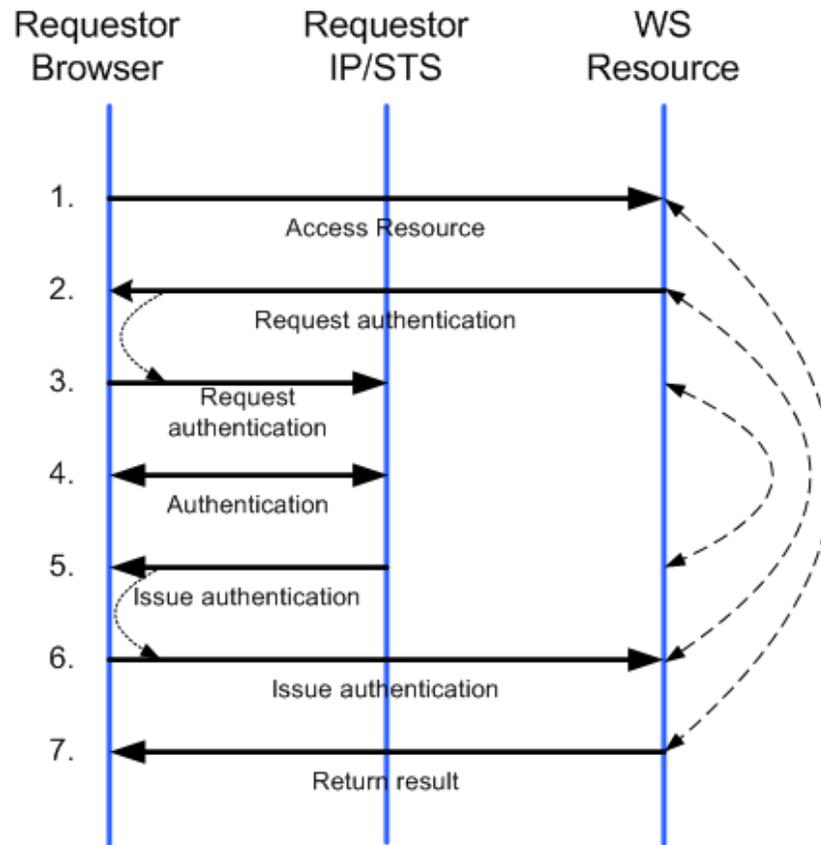
2966 The primary issue for *Web browsers* is that there is no easy way to directly issue SOAP requests.
2967 Consequently, the processing **MUST** be performed within the confines of the base HTTP 1.1 functionality
2968 (GET, POST, redirects, and cookies) and conform as closely as possible to the WS-Trust protocols for
2969 token acquisition.

2970 At a high-level, requestors are associated with an Identity Provider (IP) or Security Token Service (STS)
2971 where they authenticate themselves. At the time/point of initial authentication an artifact/cookie **MAY** be
2972 created for the requestor at their Identity Provider so that every request for a resource doesn't require
2973 requestor intervention. At other times, authentication at each request is the desired behavior.

2974 In the Web approach, there is a common pattern used when communicating with an IP/STS. In the first
2975 step, the requestor accesses the resource; the requestor is then redirected to an IP/STS if no token or
2976 cookie is supplied on the request. The requestor **MAY** be redirected to a local IP/STS operated by
2977 the resource provider. If it has not cached data indicating that the requestor has already been
2978 authenticated, a second redirection to the requestor's IP/STS will be performed. This redirection process
2979 **MAY** require prompting the user to determine the requestor's home realm. The IP/STS in the requestor's
2980 home realm generates a security token for use by the federated party. This token **MAY** be consumed
2981 directly by the resource, or it **MAY** be exchanged at the resource's IP/STS for a token consumable by the
2982 resource. In some cases the requestor's IP/STS has the requisite information cached to be able to issue
2983 a token, in other cases it must prompt the user. Note that the resource's IP/STS can be omitted if the
2984 resource is willing to consume the requestor's token directly.

2985 The figure below illustrates an example flow where there is no resource IP/STS. As depicted, all
2986 communication occurs with the standard HTTP GET and POST methods, using redirects (steps 2→3 and
2987 5→6) to automate the communication. Note that when returning non-URL content a POST is **REQUIRED**
2988 (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
2989 communication with an additional service isn't required. Note that step 3 depicts the resource redirecting
2990 directly to the requestor's IP/STS. As previously discussed, this could redirect to an IP/STS for the
2991 resource (or any number of chained IP/STS services). It might also redirect to a home realm discovery
2992 service.

2993 It should be noted that in step 4, the authentication protocol employed MAY be implementation-
2994 dependent.



2995

2996

Figure 25: Sample Browser Sign-On

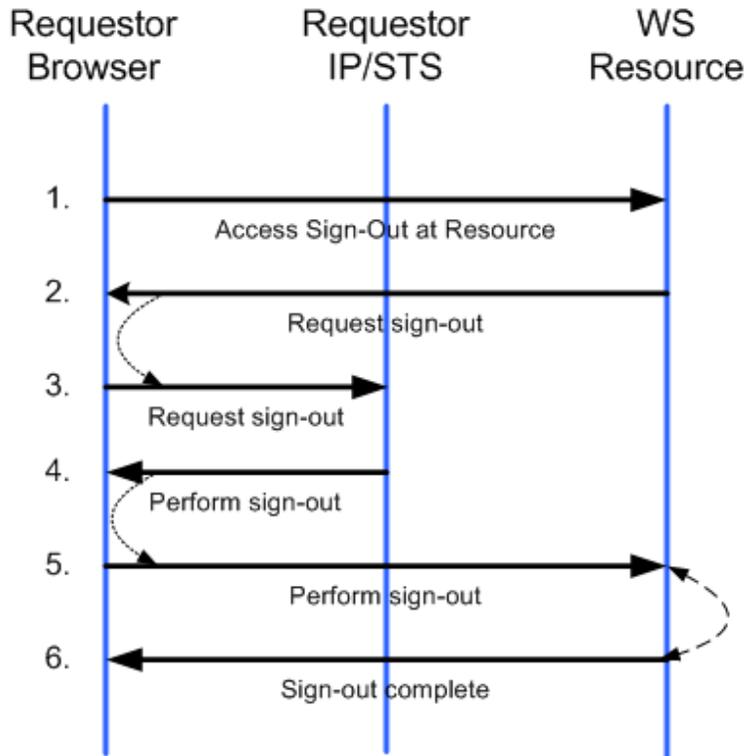
2997 13.1.2 Sign-Out

2998 For Web browsers, sign-out can be initiated by selecting the sign-out URL at a resource. In doing so, the
2999 browser will ultimately be redirected to the requestor's IP/STS indicating sign-out. Note that the browser
3000 MAY be first redirected to the resource's IP/STS and then to the requestor's IP/STS. Note that if multiple
3001 IP/STS services are used, and unaware of each other, multiple sign-outs MAY be required.

3002 The requestor's IP/STS SHOULD keep track of the realms to which it has issued tokens where cleanup
3003 may be required – specifically the IP/STS for the realms (or resources if different). When the sign-out is
3004 received at the requestor's IP/STS, it SHOULD initiate clean-up (e.g. issuing HTTP GET requests against
3005 the tracked realms indicating a sign-out cleanup is in effect or it can use the sign-out mechanism
3006 previously discussed). The exact mechanism by which this occurs is up to the IP/STS and is policy-
3007 driven. The only requirement is that a sign-out cleanup be performed at the IP/STS so that subsequent
3008 requests to the IP/STS don't use cached data.

3009 As described in section 4.2, there are two possible flows for these messages. They could be effectively
3010 chained through all the STSs involved in the session by successively redirecting the browser between
3011 each resource IP/STS and the requestor's IP/STS. Or the requestor's IP/STS can send sign-out
3012 messages to all the other STSs in parallel. The chained (sequential) approach has been found to be
3013 fragile in practice. If a resource IP/STS fails to redirect the user after cleaning up local state, or the
3014 network partitions, the sign-out notification will not reach all the resource IP/STSs involved. For this
3015 reason, compliant implementations SHOULD employ the parallel approach.

3016 When a sign-out clean-up GET is received at a realm, the realm SHOULD clean-up any cached
 3017 information and delete any associated artifacts/cookies. If requested, on completion the requestor is
 3018 redirected back to requestor's IP/STS.



3019
 3020

Figure 26: Sample Browser Sign-Out

3021 The figure above illustrates this process where a resource-specific IP/STS doesn't exist. The mechanism
 3022 illustrated use redirection in steps 2 and 4 (optional) and the general *correlation* of messages to chain the
 3023 sign-out. As previously noted there could be a resource-specific IP/STS which handles local chaining or
 3024 notification.

3025 It should be noted that as a result of the single sign-out request (steps 5 and 6), an IP/STS MAY send
 3026 sign-out messages as described in this specification.

3027 **13.1.3 Attributes**

3028 At a high-level, attribute processing uses the same mechanisms defined for security token service
 3029 requests and responses. That is, redirection is used to issue requests to attribute services and
 3030 subsequent redirection returns the results of the attribute operations. All communication occurs with the
 3031 standard HTTP 1.1 GET and POST methods using redirects to automate the communication as shown in
 3032 the example below.

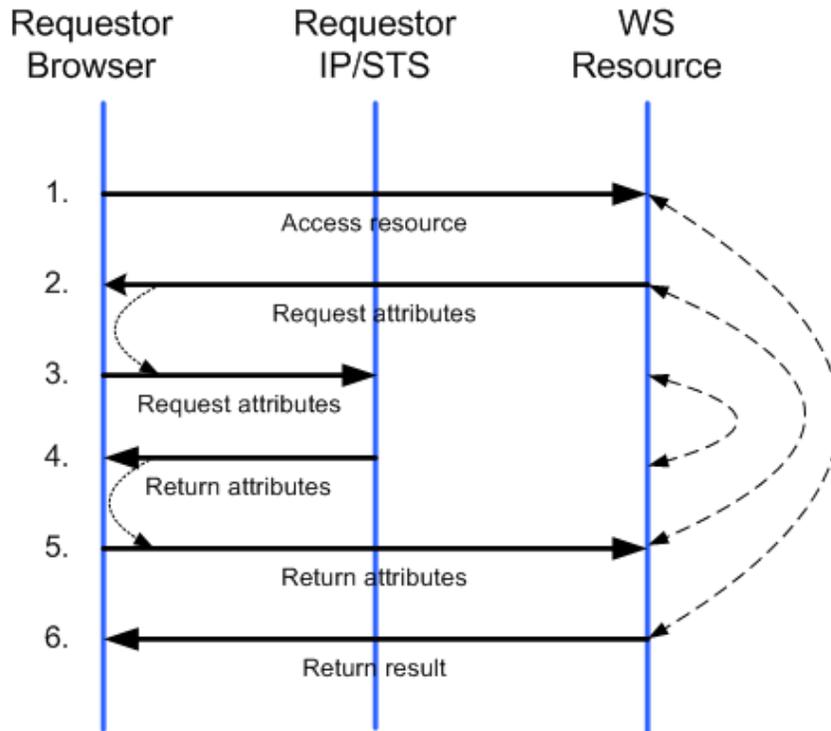


Figure 27: Sample Browser Attribute Access

3033

3034

3035 The figure above illustrates this process including calling out the redirection in steps 2 and 4 and the
 3036 general *correlation* of messages for an attribute scenario where there is no resource-specific IP/STS.

3037 As well, it should be noted that as a result of step 3 the IP/STS MAY prompt the user for approval before
 3038 proceeding to step 4.

3039 13.1.4 Pseudonyms

3040 At a high-level, pseudonym processing uses the same mechanisms defined for attribute and security
 3041 token service requests. That is, redirection is used to issue requests to pseudonym services and
 3042 subsequent redirection returns the results of the pseudonym operations. All communication occurs with
 3043 the standard HTTP GET and POST methods using redirects to automate the communication as in the
 3044 example below.

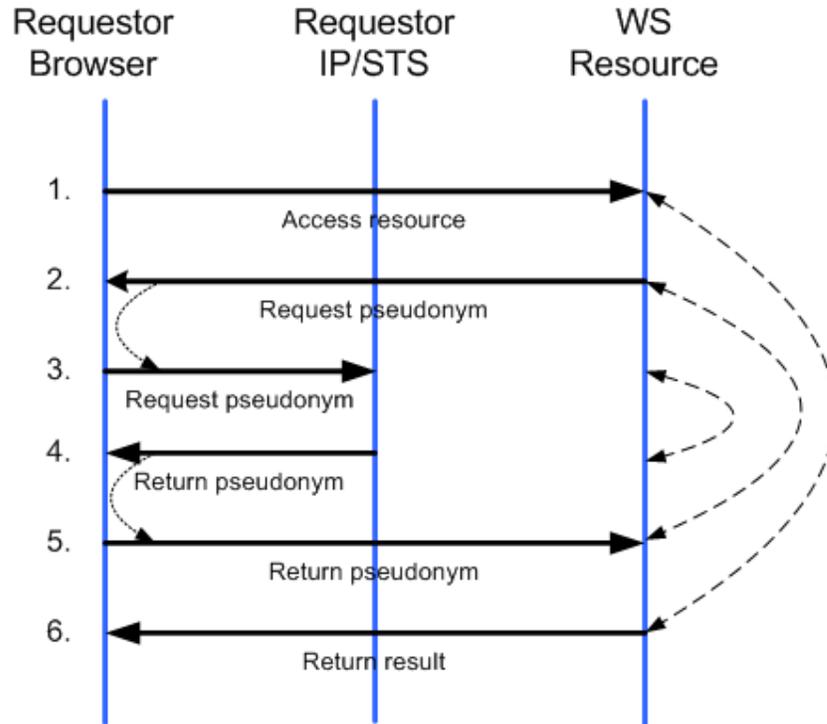


Figure 28: Sample Browser Pseudonym Access

3045

3046

3047 The figure above illustrates this process including calling out the redirection in steps 2 and 4 and the
 3048 general *correlation* of messages for an attribute scenario where there is no resource-specific IP/STS.

3049 13.1.5 Artifacts/Cookies

3050 In order to prevent requestor interaction on every request for security token, artifacts/cookies can be used
 3051 by SSO implementations as they are used today to cache state and/or authentication information or
 3052 issued tokens. However implementations MAY omit this caching if the desired behavior is to authenticate
 3053 on every request. As noted in the Security Consideration section later in this document, there are
 3054 security issues when using cookies.

3055 There are no restrictions placed on artifacts/cookie formats – they are up to each service to determine.
 3056 However, it is RECOMMENDED artifacts/cookies be encrypted or computationally hard to compromise.

3057 13.1.6 Bearer Tokens and Token References

3058 In cases where bearer tokens or references to tokens are passed it is strongly RECOMMENDED that the
 3059 messages use transport security in order to prevent attack.

3060 13.1.7 Freshness

3061 In cases where a resource requires specific authentication freshness, they can specify requirements in
 3062 their IP/STS requests, as described in the following section (see 13.2.2).

3063 13.2 HTTP Protocol Syntax

3064 This section describes the syntax of the protocols used by Web requestors. This protocol typically uses
3065 the redirection facilities of HTTP 1.1. This happens using a standard HTTP 302 error code for redirects
3066 (as illustrated below) and HTTP POST to push the forms:

```
3067 HTTP/1.1 302 Found  
3068 Location: url?parameters
```

3069 The exact parameters and form fields are described in detail in the sub-sections that follow the detailed
3070 example.

3071 In the descriptions below, some mechanisms are OPTIONAL meaning they MAY be supported. Within a
3072 mechanism, certain parameters MUST be specified while others, noted using square brackets, are
3073 OPTIONAL and MAY (or might not) be present.

3074 13.2.1 Parameters

3075 All HTTP 1.1 methods (both GET and POST) used in the redirection protocol allow query string
3076 parameters as illustrated below:

```
3077 GET url?parameters  
3078 POST url?parameters
```

3079 The GET and POST requests have required parameters and may have optional parameters depending
3080 on the operation being performed. For GET requests, these parameters are specified in the query string;
3081 for POST requests, these parameters are specified in the POST body (using the standard encoding rules
3082 for POST). The query string parameters of a POST request SHOULD be for extensibility only, and MAY
3083 be ignored by an implementation that is otherwise compliant with this specification.

3084 The following describes the parameters used for messages in this profile:

```
3085 wa=string  
3086 [wreply=URL]  
3087 [wres=URL]  
3088 [wctx=string]  
3089 [wp=URI]  
3090 [wct=timestring]  
3091 [wfed=string]  
3092 [wencoding=string]
```

3093 wa

3094 This REQUIRED parameter specifies the action to be performed. By including the action, URIs
3095 can be overloaded to perform multiple functions. For sign-in, this string MUST be "wsignin1.0".
3096 Note that this serves roughly the same purpose as the WS-Addressing Action header for the WS-
3097 Trust SOAP RST messages.

3098 wreply

3099 This OPTIONAL parameter is the URL to which responses are directed. Note that this serves
3100 roughly the same purpose as the WS-Addressing <wsa:ReplyTo> header for the WS-Trust
3101 SOAP RST messages.

3102 wres

3103 This OPTIONAL parameter is the URL for the resource accessed. This is a legacy parameter
3104 which isn't typically used. The *wrealm* parameter is typically used instead.

3105 wctx

3106 This OPTIONAL parameter is an opaque context value that MUST be returned with the issued
3107 token if it is passed in the request. Note that this serves roughly the same purpose as the WS-

3108 Trust SOAP RST @Context attribute. In order not to exceed URI length limitations, the value of
3109 this parameter should be as small as possible.

3110 wp

3111 This OPTIONAL parameter is the URL for the policy which can be obtained using an HTTP GET
3112 and identifies the policy to be used related to the action specified in "wa", but MAY have a
3113 broader scope than just the "wa". Refer to WS-Policy and WS-Trust for details on policy and
3114 trust. This attribute is only used to reference policy documents. Note that this serves roughly the
3115 same purpose as the Policy element in the WS-Trust SOAP RST messages.

3116 wct

3117 This OPTIONAL parameter indicates the current time at the sender for ensuring freshness. This
3118 parameter is the string encoding of time using the XML Schema datetime time using UTC
3119 notation. Note that this serves roughly the same purpose as the WS-Security Timestamp
3120 elements in the Security headers of the SOAP RST messages.

3121 wfed

3122 This OPTIONAL parameter indicates the federation context in which the request is made. This is
3123 equivalent to the `FederationId` parameter in the RST message.

3124 wencoding

3125 This OPTIONAL parameter indicates the encoding style to be used for XML parameter content. If
3126 not specified the default behavior is to use standard URL encoding rules. This specification only
3127 defines one other alternative, `base64url` as defined in section 5 of [RFC 4648]. Support for
3128 alternate encodings is expressed by assertions under the WebBinding assertion defined in this
3129 specification.

3130 Note that any values specified in parameters are subject to encoding as specified in the HTTP 1.1
3131 specification.

3132 When an HTTP POST is used, any of the query strings can be specified in the form contents using the
3133 same name. Note that in this profile form values take precedence over URL parameters.

3134 Parameterization is extensible so that cooperating parties can exchange additional information in
3135 parameters based on agreements or policy.

3136 13.2.2 Requesting Security Tokens

3137 The HTTP requests to an Identity Provider or security token service use a common syntax based on
3138 HTTP forms. Requests typically arrive using the HTTP GET method as illustrated below but MAY be
3139 issued using a POST method:

```
3140 GET resourceSTS?parameters HTTP/1.1  
3141 POST resourceSTS?parameters HTTP/1.1
```

3142 The parameters described in the previous section (wa, wreply, wres, wctx, wp, wct) apply to the token
3143 request. The additional parameters described below also apply. Note that any values specified in forms
3144 are subject to encoding as described in the HTTP 1.1 specification.

3145 The following describes the additional parameters used for a token request:

```
3146 wrealm=string  
3147 [wfresh=freshness]  
3148 [wauth=uri]  
3149 [wreq=xml]
```

3150 wrealm

3151 This REQUIRED parameter is the URI of the requesting realm. The wrealm SHOULD be the
3152 security realm of the resource in which nobody (except the resource or authorized delegates) can

3153 control URLs. Note that this serves roughly the same purpose as the AppliesTo element in the
3154 WS-Trust SOAP RST messages.

3155 wfresh

3156 This OPTIONAL parameter indicates the freshness requirements. If specified, this indicates the
3157 desired maximum age of authentication specified in minutes. An IP/STS SHOULD NOT issue a
3158 token with a longer lifetime. If specified as "0" it indicates a request for the IP/STS to re-prompt
3159 the user for authentication before issuing the token. Note that this serves roughly the same
3160 purpose as the Freshness element in the WS-Trust SOAP RST messages.

3161 wauth

3162 This OPTIONAL parameter indicates the REQUIRED authentication level. Note that this
3163 parameter uses the same URIs and is equivalent to the `wst:AuthenticationType` element in
3164 the WS-Trust SOAP RST messages.

3165 wreq

3166 This OPTIONAL parameter specifies a token request using either a
3167 `<wst:RequestSecurityToken>` element or a full request message as described in WS-Trust.
3168 If this parameter is not specified, it is assumed that the responding service *knows* the correct type
3169 of token to return. Note that this can contain the same RST payload as used in WS-Trust RST
3170 messages.

3171 To complete the protocol for requesting a token, it is necessary to redirect the Web requestor from the
3172 resource, or its local IP/STS, to the requestor's IP/STS. Determining the location of this IP/STS is
3173 frequently referred to as Home Realm Discovery; that is, determining the realm which manages the
3174 requestor's identity and thus where its IP/STS is located. This frequently involves interaction with the
3175 user (see section 13.5 for additional discussion). There are situations – particularly when users only
3176 access resources via portals and never directly via bookmarked URLs – when it can be advantageous to
3177 include the requestor's home realm in the request to avoid the requirement for human interaction. The
3178 following parameter MAY be specified for this purpose.

3179 `[whr=string]`

3180 whr

3181 This OPTIONAL parameter indicates the account partner realm of the client. This parameter is
3182 used to indicate the IP/STS address for the requestor. This may be specified directly as a URL or
3183 indirectly as an identifier (e.g. urn: or uuid:). In the case of an identifier the recipient is expected
3184 to know how to translate this (or get it translated) to a URL. When the *whr* parameter is used, the
3185 resource, or its local IP/STS, typically removes the parameter and writes a cookie to the client
3186 browser to remember this setting for future requests. Then, the request proceeds in the same
3187 way as if it had not been provided. Note that this serves roughly the same purpose as federation
3188 metadata for discovering IP/STS locations previously discussed.

3189 In the event that the XML request cannot be passed in the form (due to size or other considerations), the
3190 following parameter MAY be specified and the form made available by reference:

3191 `wreqptr=url`

3192 wreqptr

3193 This OPTIONAL parameter specifies a URL for where to find the request expressed as a
3194 `<wst:RequestSecurityToken>` element. Note that this does not have a WS-Trust parallel.
3195 The *wreqptr* parameter MUST NOT be included in a token request if *wreq* is present.

3196 When using *wreqptr* it is strongly RECOMMENDED that the provider of the *wreqptr* data authenticate the
3197 data to the consumer (relying party) in some way and that the provider authenticate consumers

3198 requesting the wreqptr data. If the wreqptr data is sensitive the provider SHOULD consider ensuring
3199 confidentiality of the data transfer.
3200 The RST is logically constructed to process the request. If one is specified (either directly via wreq or
3201 indirectly via wreqptr) it is the authoritative source for parameter information. That is, parameters outside
3202 of the RST (e.g. wfresh, wtrealm, ...) are used to construct an RST if the RST is not present or if the
3203 corresponding RST values are not present.

3204 13.2.3 Returning Security Tokens

3205 Security tokens are returned by passing an HTTP form. To return the tokens, this profile embeds a
3206 <wst:RequestSecurityTokenResponse> element as specified in [WS-Trust].

```
3207 POST resourceURI?parameters HTTP/1.1  
3208 GET resourceURI?parameters HTTP/1.1
```

3209 In many cases the IP/STS to whom the request is being made, will prompt the requestor for information or
3210 for confirmation of the receipt of the token. As a result, the IP/STS can return an HTTP form to the
3211 requestor who then submits the form using an HTTP POST method. This allows the IP/STS to return
3212 security token request responses in the body rather than embedded in the limited URL query string.
3213 However, in some circumstances interaction with the requestor may not be required (e.g. cached
3214 information). In these circumstances the IP/STS have several options:

- 3215 1. Use a form anyway to confirm the action
- 3216 2. Return a form with script to automate and instructions for the requestor in the event that scripting
3217 has been disabled
- 3218 3. Use HTTP GET and return a pointer to the token request response (unless it is small enough to fit
3219 inside the query string)

3220 This specification RECOMMENDS using the POST method as the GET method requires additional state
3221 to be maintained and complicates the cleanup process whereas the POST method carries the state inside
3222 the method.

3223 Note that when using the POST method, any values specified in parameters are subject to encoding as
3224 described in the HTTP 1.1 specification. The standard parameters apply to returning tokens as do the
3225 following additional form parameters:

```
3226 wresult=xml  
3227 [wctx=string]
```

3228 wresult

3229 This REQUIRED parameter specifies the result of the token issuance. This can take the form of
3230 the <wst:RequestSecurityTokenResponse> element or
3231 <wst:RequestSecurityTokenResponseCollection> element, a SOAP security token
3232 request response (that is, a <S:Envelope>) as detailed in WS-Trust, or a SOAP <S:Fault>
3233 element. This carries the same content as a WS-Trust RSTR element (or even the actual SOAP
3234 Envelope containing the RSTR element).

3235 wctx

3236 This OPTIONAL parameter specifies the context information (if any) passed in with the request
3237 and typically represents context from the original request.

3238 In the event that the token/result cannot be passed in the form, the following parameter MAY be specified:

```
3239 wresultptr=url
```

3240 wresultptr

3241 This parameter specifies a URL to which an HTTP GET can be issued. The result is a document
3242 of type `text/xml` that contains the issuance result. This can either be the
3243 `<wst:RequestSecurityTokenResponse>` element, the
3244 `<wst:RequestSecurityTokenResponseCollection>` element, a SOAP response, or a
3245 SOAP `<S:Fault>` element. Note that this serves roughly the same purpose as the WS-
3246 ReferenceToken mechanism previously discussed (although this is used for the full response not
3247 just the token).

3248 13.2.4 Sign-Out Request Syntax

3249 This section describes how sign-out requests are formed and redirected by Web requestors. For
3250 modularity, it should be noted that support for sign-out is OPTIONAL.

3251 Sign-out can be initiated by a client at one of four points in the system:

- 3252 1. A Relying Party application server
- 3253 2. A Relying Party STS
- 3254 3. An application server local to the Identity Provider
- 3255 4. The Identity Provider STS

3256 For the first three use cases, the requestor's client must be redirected to the Identity Provider STS where
3257 the current session originated. This STS is required to send clean-up messages to all Relying Party STSs
3258 and any local applications for which the IP STS has issued security tokens for the requestor's current
3259 session. How the STS tracks this state for the requestor is implementation specific and outside the scope
3260 of this specification.

3261 As can be seen, for passive requestors the sign-out process is divided into two separate phases, referred
3262 to as sign-out and clean-up. Two different messages are used to ensure that all components of the
3263 system understand which phase is in effect to ensure that the requestor's sign-out request is processed
3264 correctly.

3265 13.2.4.1 Sign-out Message Syntax

3266

3267 The following describes the parameters used for the sign-out request (note that this parallels the sign-out
3268 SOAP message previously discussed):

```
3269 wa=string  
3270 wreply=URL
```

3271 wa

3272 This REQUIRED parameter specifies the action to be performed. By including the action, URIs
3273 can be overloaded to perform multiple functions. For sign-out, this string MUST be "wsignout1.0".

3274

3275 wreply

3276 This OPTIONAL parameter specifies the URL to return to once clean-up (sign-out) is complete. If
3277 this parameter is not specified, then after cleanup the GET completes by returning any realm-
3278 specific data such as a string indicating cleanup is complete for the realm.

3279 13.2.4.2 Clean-up Message Syntax

3280 The following describes the parameters used for the clean-up phase of a sign-out
3281 request:

```
3282 wa=string
3283 wreply=URL
```

3284 wa

3285 This **required** parameter specifies the action to be performed. By including the action, URIs can
3286 be overloaded to perform multiple functions. For the clean-up phase of a sign-out request, this
3287 string **MUST** be "wsignoutcleanup1.0".

3288 wreply

3289 This **optional** parameter specifies the URL to return to once clean-up is complete. If this
3290 parameter is not specified, then after cleanup the GET **MAY** complete by returning any realm-
3291 specific data such as a string indicating cleanup is complete for the realm.

3292

3293 13.2.5 Attribute Request Syntax

3294 This section describes how attribute requests are formed and redirected by Web requestors. For
3295 modularity, it should be noted that support for attributes is **OPTIONAL**. Additionally it should be noted
3296 that security considerations may apply. While the structure described here **MAY** be used with an attribute
3297 service supporting Web clients, the actual attribute request and response XML syntax is undefined and
3298 specific to the attribute store.

3299 The following describes the valid parameters used within attributes requests:

```
3300 wa=string
3301 [wreply=URL]
3302 [wrealm=URL]
3303 wattr=xml-attribute-request
3304 wattrptr=URL
3305 wresult=xml-result
3306 wresultptr=URL
```

3307 wa

3308 This **REQUIRED** parameter specifies the action to be performed. By including the action, URIs
3309 can be overloaded to perform multiple functions. For attribute requests, this string **MUST** be
3310 "wattr1.0".

3311 wreply

3312 This **OPTIONAL** parameter specifies the URL to return to when the attribute result is complete.

3313 wattr

3314 This **OPTIONAL** parameter specifies the attribute request. The syntax is specific to the attribute
3315 store being used and is not mandated by this specification. This attribute is only present on the
3316 request.

3317 wattrptr

3318 This **OPTIONAL** parameter specifies URL where the request can be obtained.

3319 wresult

3320 This **OPTIONAL** parameter specifies the result as defined by the attribute store and is not
3321 mandated by this specification. This attribute is only present on the responses.

3322 wresultptr

3323 This **OPTIONAL** parameter specifies URL where the result can be obtained.

3324 13.2.6 Pseudonym Request Syntax

3325 This section describes how pseudonym requests are formed and redirected by Web requestors. For
3326 modularity, it should be noted that support for pseudonyms is also OPTIONAL. As well, it should be
3327 noted that security considerations may apply.

3328 The following describes the valid parameters used within pseudonym requests (note that this parallels the
3329 pseudonym messages previously discussed):

```
3330 wa=string  
3331 [wreply=URL]  
3332 [wrealm=URL]  
3333 wpseudo=xml-pseudonym-request  
3334 wpseudoptr=URL  
3335 wresult=xml-result  
3336 wresultptr=URL
```

3337 wa

3338 This REQUIRED parameter specifies the action to be performed. By including the action, URIs
3339 can be overloaded to perform multiple functions. For pseudonym requests, this string MUST be
3340 "wpseudo1.0".

3341 wreply

3342 This OPTIONAL parameter specifies the URL to return to when the pseudonym result is
3343 complete.

3344 wpseudo

3345 This OPTIONAL parameter specifies the pseudonym request and either contains a SOAP
3346 envelope or a pseudonym request, such as a WS-Transfer/WS-ResourceTransfer <Get>. This
3347 attribute is only present on the request.

3348 wpseudoptr

3349 This OPTIONAL parameter specifies URL from which the request element can be obtained.

3350 wresult

3351 This OPTIONAL parameter specifies the result as either a SOAP envelope or a pseudonym
3352 response. This attribute is only present on the responses.

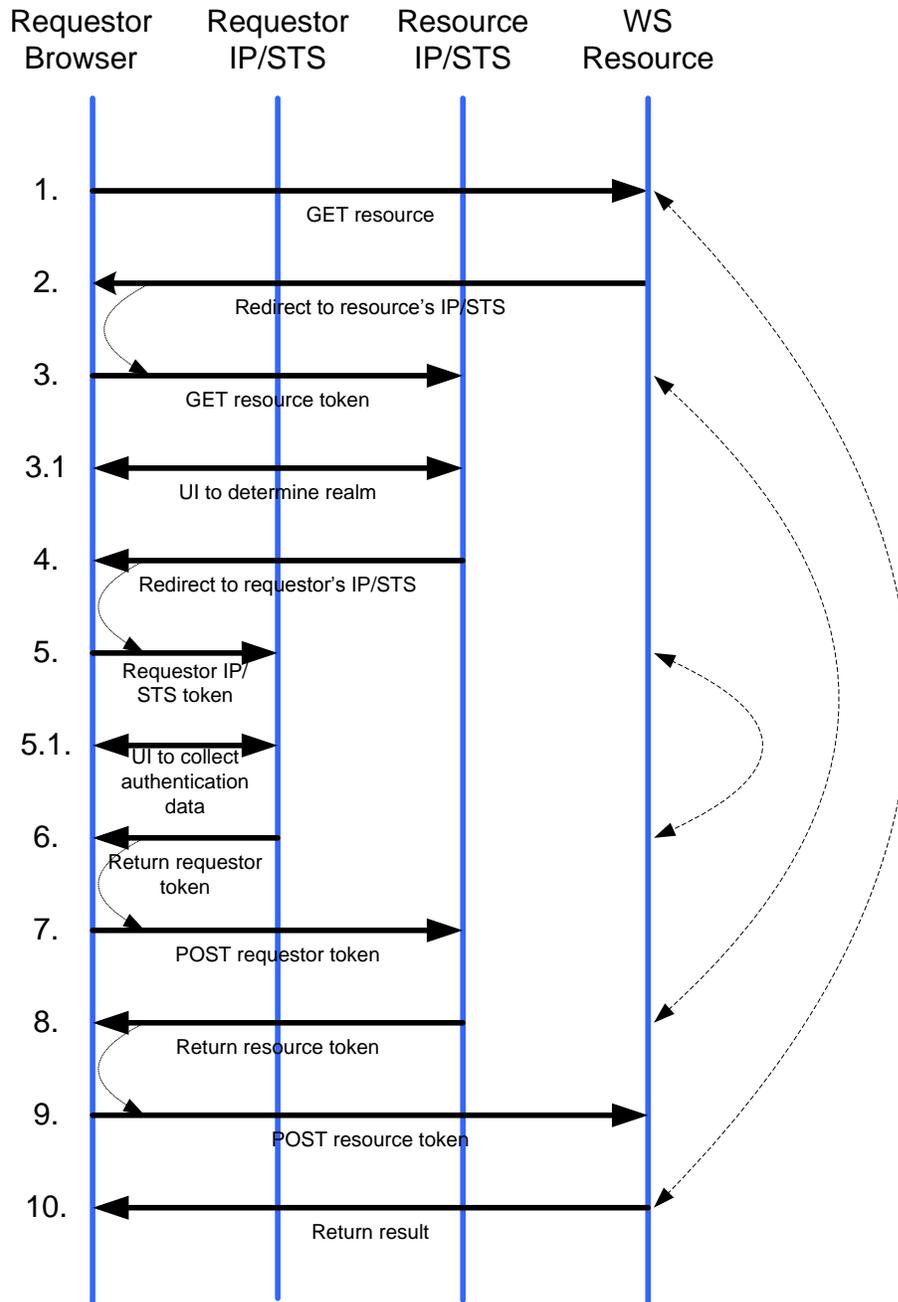
3353 wresultptr

3354 This optional OPTIONAL parameter specifies URL from which the result element can be
3355 obtained.

3356 13.3 Detailed Example of Web Requester Syntax

3357 This section provides a detailed example of the protocol defined in this specification. The exact flow for
3358 Web sign-in scenarios can vary significantly; however, the following diagram and description depict a
3359 *common* or basic sequence of events.

3360 In this scenario, the user at a requestor browser is attempting to access a resource which requires
3361 security authentication to be validated by the resource's security token service. In this example there is a
3362 resource-specific IP/STS.



3363

3364

Figure 29: Details Sample Browser Sign-In

3365

Simple Scenario:

3366

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.

3367

3368

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

3369

3370

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

3371

3372

3373 throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard
3374 HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method
3375 described in step 6).

3376 It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of
3377 the information.

3378 **Step 3:** Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor
3379 realm MAY be cached in an artifact/cookie from an earlier exchange, it MAY be known to or fixed by the
3380 resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

3381 **Step 3.1:** This is an OPTIONAL step. If the resource IP/STS cannot determine the requestor's realm,
3382 then the IP/STS MAY prompt the requestor for realm information.

3383 **Step 4:** The resource IP/STS redirects to the requestor's IP/STS in order to validate the requestor.
3384 Typically, this is done using a HTTP 302 redirect.

3385 As in step 2, additional information MAY be passed to reflect the agreement between the two IP/STS's,
3386 and this request for the token MAY be done using a POST method (see syntax for details).

3387 The requestor IP/STS SHOULD provide information confidentiality or use HTTP/S or some other
3388 transport-level security mechanism.

3389 **Step 5:** The requestor's IP/STS now authenticates the requestor to establish a sign in.

3390 **Step 5.1:** Validation of the requestor MAY involve displaying some UI in this OPTIONAL step.

3391 **Step 6:** Once requestor information has been successfully validated, a security token response (RSTR) is
3392 formatted and sent to the resource IP/STS.

3393 Processing continues at the resource IP/STS via a redirect.

3394 While an IP/STS MAY choose to return a pointer to token information using `wresultptr`, it is
3395 RECOMMENDED that, whenever possible to return the security token (RSTR) using a POST method to
3396 reduce the number of overall messages. This MAY be done using requestor-side scripting. The exact
3397 syntax is described in Appendix I.

3398 **Step 7:** Resource's IP/STS receives and validates the requestor's security token (RSTR).

3399 **Step 8:** The resource's IP/STS performs a federated authentication/authorization check (validation
3400 against policy). After a successful check, the resource's IP/STS can issue a security token for the
3401 resource. The resource IP/STS redirects to the resource.

3402 It should be noted that the OPTIONAL `wctx` parameter specifies the opaque context information (if any)
3403 passed in with the original request and is echoed back here. This mechanism is an optional way for the
3404 IP/STS to have state returned to it.

3405 At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the
3406 requestor (which likely includes the requestor's realm).

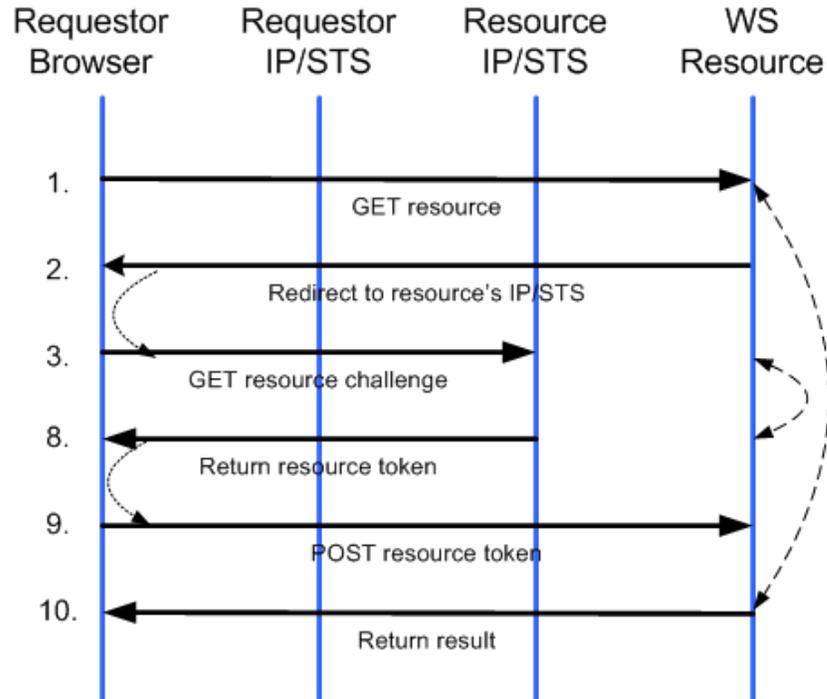
3407 **Step 9:** The resource receives the security token (RSTR) from the resource IP/STS. On successful
3408 validation the resource processes the request (per policy).

3409 The security token SHOULD be passed using an HTML POST using the syntax previously described.

3410 **Step 10:** The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it
3411 returns the result of the resource request.

3412

3413 **Optimized Scenario:**



3414

3415

Figure 30: Optimized Sample Browser Sign-In

3416 This scenario assumes that an initial federated flow has occurred. Note that many legs of the initial flow
 3417 MAY be eliminated due to the presence of artifacts/cookies. For readability, the similar steps are
 3418 numbered consistently with the previous non-optimized example.

3419 **Step 1:** The requestor browser accesses a resource, typically using the HTTP GET method.

3420 **Step 2:** At the resource, the requestor's request is redirected to the IP/STS associated with the target
 3421 resource. The redirected URL MAY contain additional information reflecting agreements which the
 3422 resource and its IP/STS have established; however, this (redirection target) URL MUST be used
 3423 throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard
 3424 HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method
 3425 described in step 6).

3426 It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of
 3427 the information.

3428 **Step 3:** Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor
 3429 realm could be cached in an artifact/cookie from an earlier exchange, it could be known to or fixed by the
 3430 resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

3431 **Step 8:** The resource's IP/STS performs a federated authentication/authorization check (validation
 3432 against policy). After a successful check, the resource's IP/STS can issue a security token for the
 3433 resource. The resource IP/STS redirects to the resource.

3434 It should be noted that the OPTIONAL `wctx` parameter specifies the opaque context information (if any)
 3435 passed in with the original request and is echoed back here. This mechanism is an optional way for the
 3436 IP/STS to have state returned to it.

3437 At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the
 3438 requestor (which likely includes the requestor's realm).

3439 **Step 9:** The resource receives the security token (RSTR) from the resource IP/STS. On successful
3440 validation the resource processes the request (per policy).
3441 The security token SHOULD be passed using an HTML POST using the syntax previously described.
3442 **Step 10:** The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it
3443 returns the result of the resource request.

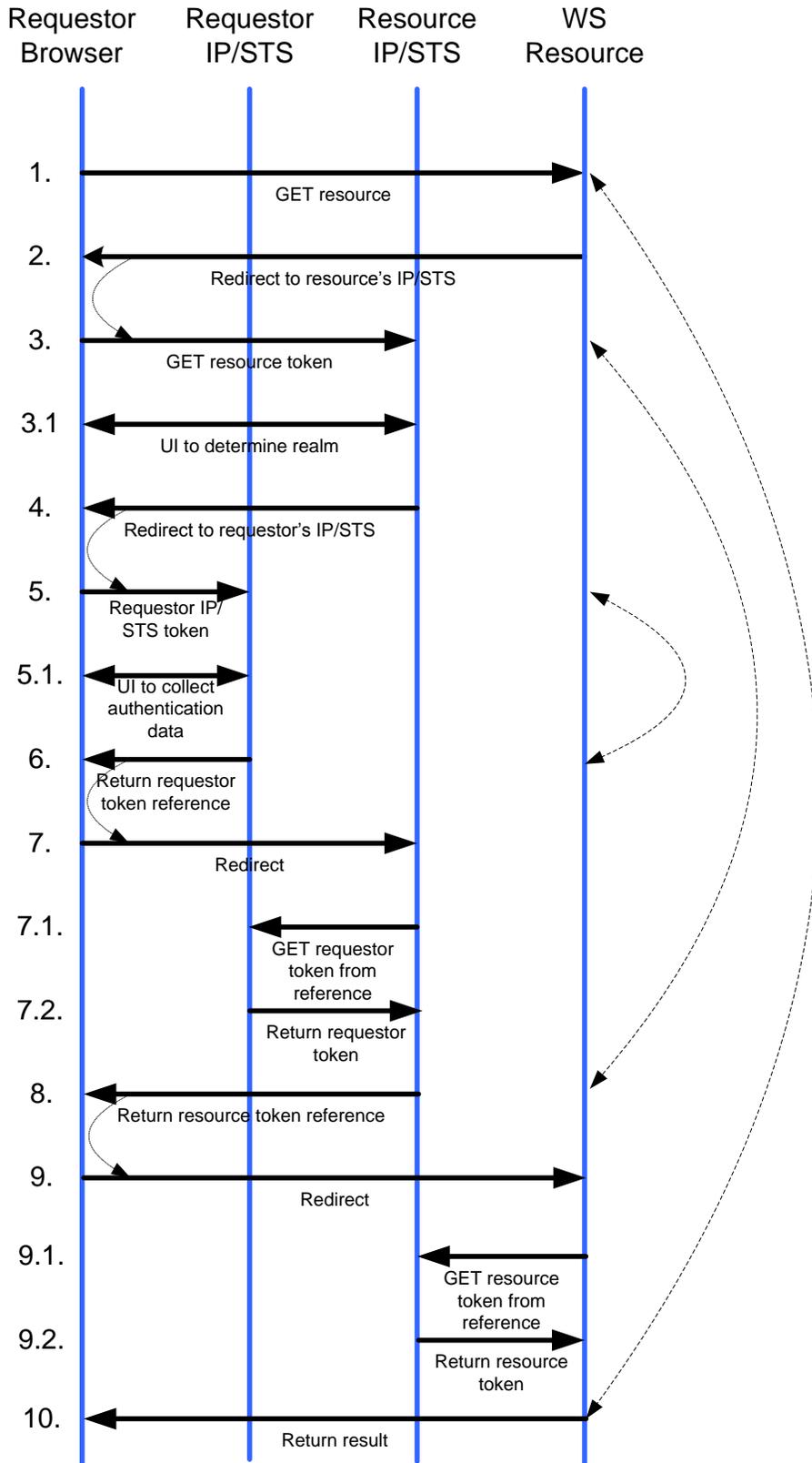
3444 **13.4 Request and Result References**

3445 The previous example illustrates a common form of messaging when passing WS-Trust messages via a
3446 simple Web browser. However, in some scenarios it is undesirable to use POST messages and carry the
3447 full details within the messages (e.g. when redirecting through wireless or mobile devices). In such cases
3448 requests and responses can be referenced via a URL and all messages passed as part of the query
3449 strings (or inside small POSTs).

3450 Request references are specified via *wreqptr* and typically specify a `<wst:RequestSecurityToken>`
3451 element that can be obtained by issuing a HTTP GET against the specified URL. Response references
3452 are specified via *wresultptr* and typically specify a `<wst:RequestSecurityTokenResponse>` or
3453 `<wst:RequestSecurityTokenResponseCollection>` element that can be obtained by issuing a
3454 HTTP GET against the specified URL.

3455 This section provides a detailed example of the use of references with the protocol defined in this
3456 specification. The exact flow for Web sign-in scenarios can vary significantly; however, the following
3457 diagram and description depict a *common* or basic sequence of events. Note that this example only
3458 illustrates result reference not request references and makes use of a resource-specific IP/STS.

3459 In this scenario, the user at a requestor browser is attempting to access a resource which requires
3460 security authentication to be validated by the resource's security token service.



3461

3462

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

3463 **Step 1:** The requestor browser accesses a resource, typically using the HTTP GET method.

3464 **Step 2:** At the resource, the requestor's request is redirected to the IP/STS associated with the target
3465 resource. The redirected URL MAY contain additional information reflecting agreements which the
3466 resource and its IP/STS have established; however, this (redirection target) URL MUST be used
3467 throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard
3468 HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method
3469 described in step 6).

3470 It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of
3471 the information.

3472 **Step 3:** Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor
3473 realm could be cached in an artifact/cookie from an earlier exchange, it could be known to or fixed by the
3474 resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

3475 **Step 3.1:** This is an OPTIONAL step. If the resource IP/STS cannot determine the requestor's realm,
3476 then the IP/STS MAY prompt the requestor for realm information.

3477 **Step 4:** The resource IP/STS redirects to the requestor's IP/STS in order to validate the requestor.
3478 Typically, this is done using a HTTP 302 redirect.

3479 As in step 2, additional information MAY be passed to reflect the agreement between the two IP/STS's,
3480 and this request for the token MAY be done using a POST method (see syntax for details).

3481 The requestor IP/STS SHOULD provide information confidentiality or use HTTP/S or some other
3482 transport-level security mechanism.

3483 **Step 5:** The requestor's IP/STS now authenticates the requestor to establish a sign in.

3484 **Step 5.1:** Validation of the requestor MAY involve displaying some UI in this OPTIONAL step.

3485 **Step 6:** Once requestor information has been successfully validated, a security token response (RSTR) is
3486 formatted and sent to the resource IP/STS.

3487 Processing continues at the resource IP/STS via a redirect.

3488 **Step 7:** Resource's IP/STS receives and validates the requestor's security token (RSTR).

3489 **Step 7.1:** The Resource IP/STS issues a GET to the Requestor IP/STS to obtain the actual RSTR.

3490 **Step 7.2:** The Requestor IP/STS responds to the GET and returns the actual RSTR.

3491 **Step 8:** The resource's IP/STS performs a federated authentication/authorization check (validation
3492 against policy). After a successful check, the resource's IP/STS can issue a security token for the
3493 resource. The resource IP/STS redirects to the resource.

3494 It should be noted that the OPTIONAL `wctx` parameter specifies the opaque context information (if any)
3495 passed in with the original request and is echoed back here. This mechanism is an optional way for the
3496 IP/STS to have state returned to it.

3497 At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the
3498 requestor (which likely includes the requestor's realm).

3499 **Step 9:** The resource receives the security token (RSTR) from the resource IP/STS. On successful
3500 validation the resource processes the request (per policy).

3501 The security token SHOULD be passed using an HTML POST using the syntax previously described.

3502 **Step 9.1:** The Resource issues a GET to the Resource IP/STS to obtain the actual RSTR.

3503 **Step 9.2:** The Resource IP/STS responds to the GET and returns the actual RSTR.

3504 **Step 10:** The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it
3505 returns the result of the resource request.

3506 **13.5 Home Realm Discovery**

3507 In the protocol previously described the resource or the resource's IP/STS must determine the IP/STS for
3508 the requestor and re-direct to obtain an identity token. After this is done, the information can be cached in
3509 a cookie (or by whatever means is desired).

3510 There is no normative way of discovering the *home realm* of the requestor, however, the following
3511 mechanisms are common methods:

- 3512 • *Fixed* – The home realm is fixed or known
- 3513 • *Requestor IP* – The home realm is determined using the requestor's IP address
- 3514 • *Prompt* – The user is prompted (typically using a Web page)
- 3515 • *Discovery Service* – A service is used to determine the home realm
- 3516 • *Shared Cookie* – A shared cookie from a shared domain is used (out of scope)

3517 The first three mechanisms are well understood, the *Discovery Service* is discussed next, and the cookie
3518 mechanism is outside the scope of this document.

3519 **13.5.1 Discovery Service**

3520 The *Home Realm Discovery Service* is a Web-based service that, through implementation-specific
3521 methods MAY be able to determine a requestor's home realm without user interaction.

3522 A resource or resource IP/STS MAY redirect to a discovery service to attempt to determine the home
3523 realm without prompting the user. The discovery service MUST redirect back to the URL specified by the
3524 *wreply* parameter. If the context parameter is specified it MUST also be specified. If the discovery
3525 service was able to determine the home realm, it is returned using the *whr* parameter defined in section
3526 13.2.2. This parameter contains a URI which identifies the home realm of the user. This SHOULD be the
3527 same URI that the user's realm uses for the *wrealm* parameter when it makes token requests to other
3528 federated partners. This value can be used to lookup the URL for the user's IP/STS for properly
3529 redirecting the token request.

3530 If the discovery service is unable to determine the home realm then the *whr* parameter is not specified
3531 and the home realm must be discovered by other means.

3532 **13.6 Minimum Requirements**

3533 For the purposes of interoperability of federated Web Single Sign-on, this sub-section defines a subset of
3534 the exchanges defined in this chapter which MUST be supported by all Web-enabled requestors and
3535 services. Optional aspects are optional for both clients and services.

3536 The scenario and diagram(s) in section 13.3 illustrates the core Sign-On messages between two
3537 federated realms. This is the center of the interoperability subset described below.

3538 **13.6.1 Requesting Security Tokens**

3539 The focus of these requirements is on the message exchange between the requestor IP/STS and the
3540 resource IP/STS. Thus, to conform to this specification, messages 1, 4, 7 & 10 MUST be supported
3541 (again refer to the figure and steps in section 13.3). All other message exchanges are implementation
3542 specific and are only provided here for guidance.

3543 A security token is requested via SignIn message in step 2 of the diagram. Message 3 arrives via HTTP
3544 GET and is protected by SSL/TLS. The parameters are encoded in a query string as specified in section
3545 13.2. The message will contain parameters as detailed below. Parameters enclosed in brackets are
3546 OPTIONAL.

3547

```
3548 wa=wsignin1.0
3549 wtrealm=resource realm URI
3550 [wreply=Resource IP/STS Url]
3551 [wctx=anything]
3552 [wct=ISO8601 UTC]
```

3553

3554 The REQUIRED *wa* field is common to all SignIn messages and is fixed.

3555 The REQUIRED *wtrealm* field MUST contain a URI that the *Resource IP/STS* and *Requestor IP/STS*
3556 have agreed to use to identify the realm of *Resource IP/STS* in messages to *Requestor IP/STS*.

3557 The OPTIONAL *wreply* field specifies the URL to which this message's response will be POSTed (see
3558 Returning Security Tokens).

3559 The OPTIONAL *wctx* field is provided for *Resource IP/STS*'s use and MUST be returned by *Requestor*
3560 *IP/STS* unchanged.

3561 The OPTIONAL *wct* field, if present, MUST contain the current time in UTC using the ISO8601 format
3562 (e.g. "2003-04-30T22:47:20Z"). This field MAY not be available if the requestor is coming via a portal link.
3563 Individual implementations of *Requestor IP/STS* MAY require this field to be present.

3564 Other options MAY be specified but are not required to be supported.

3565 13.6.2 Returning Security Tokens

3566 A security token is returned in response to successful Web SignIn messages, as described in the
3567 example protocol message flow in section 13.3. Security tokens are returned to the requestor and
3568 SHOULD be transmitted to a Resource Provider via HTTP POST and be protected by SSL/TLS, as
3569 depicted in steps 6-7 and 9-10 of figure 29. Optionally, the token MAY be returned using the *wresultptr*
3570 parameter. Encoding of the parameters in the POST body MUST be supported. The parameters to the
3571 message MAY be encoded in the query string if *wresultptr* is being used. The message will contain
3572 parameters as detailed below. Parameters enclosed in brackets are OPTIONAL.

3573

```
3574 wa=wsignin1.0
3575 wresult=RequestSecurityTokenResponse
3576 [wctx=wctx from the request]
3577 [wresultptr=URL]
```

3578

3579 The REQUIRED *wa* field is common to all SignIn messages and is fixed.

3580 The REQUIRED *wresult* field MUST contain a `<wst:RequestSecurityTokenResponse>` element, as
3581 detailed below.

3582 The OPTIONAL *wctx* field MUST be identical to the *wctx* field from the incoming SignIn message that
3583 evoked this response.

3584 The OPTIONAL *wresultptr* field provides a pointer to the resulting
3585 `<wst:RequestSecurityTokenResponse>` element, as detailed below.

3586 13.6.3 Details of the RequestSecurityTokenResponse element

3587 The `<wst:RequestSecurityTokenResponse>` element that is included as the *wresult* field in the
3588 SignIn response MUST contain a `<wst:RequestedSecurityToken>` element. Support for SAML
3589 assertions MUST be provided but another token format MAY be used (depending on policy).

3590 The `<wst:RequestSecurityTokenResponse>` element MAY include a *wsp:AppliesTo* /
3591 *wsa:EndpointReference* / *wsa:Address* element that specifies the Resource Realm URI. Note that
3592 this data MUST be consistent with similar data present in security tokens (if any is present) – for example

3593 it must duplicate the information in the signed token's *saml:Audience* element when SAML security
3594 tokens are returned.

3595 **13.6.4 Details of the Returned Security Token Signature**

3596 It MUST be possible to return signed security tokens, but unsecured tokens MAY be returned. Signed
3597 security tokens SHOULD contain an enveloped signature to prevent tampering but MAY use alternative
3598 methods if the security token format allows for specialized augmentation of the token. The signature
3599 SHOULD be performed over canonicalized XML [XML-C14N] (failure to do so MAY result in non-verifiable
3600 security tokens). The signature SHOULD be produced using the *Requestor STS* private key, which
3601 SHOULD correspond to either a security token included as part of the response or pre-established with
3602 the requestor. Note that in the above example the certificate is included directly in KeyInfo (via the
3603 X509Data element [WSS:X509Token]). This is the RECOMMENDED approach.

3604 When used, the X509SKI element contains the base64 encoded plain (i.e., non-DER-encoded) value of
3605 an X509 V.3 SubjectKeyIdentifier extension. If the SubjectKeyIdentifier field is not present in the
3606 certificate, the certificate itself MUST be included directly in KeyInfo (see the above example).

3607 Note that typically the returned security token is unencrypted (The entire RSTR is sent over SSL3.0/TLS
3608 [HTTPS]) but it MAY be encrypted in specialized scenarios.

3609 Take care to include appropriate transforms in *Signature/Reference/Transforms*. For example, all SAML
3610 tokens [WSS:SAMLTokenProfile] following the rules above MUST contain the enveloped signature and
3611 EXCLUSIVE canonicalization transforms.

3612 **13.6.5 Request and Response References**

3613 If the *wreqptr* or *wresultptr* parameters are supported, it MUST be possible to pass
3614 `<wst:RequestSecurityToken>` in the *wreqptr* and either
3615 `<wst:RequestSecurityTokenResponse>` or
3616 `<wst:RequestSecurityTokenResponseCollection>` in *wresultptr*. Other values MAY (but are not
3617 required) to be supported.

3618 14 Additional Policy Assertions

3619 This specification defines the following assertions for use with [WS-Policy] and [WS-SecurityPolicy].

3620 14.1 RequireReferenceToken Assertion

3621 This element represents a requirement to include a ReferenceToken (as described previously in this
3622 specification). The default version of this token is the version described in this document.

3623 The syntax is as follows:

```
3624 <fed:RequireReferenceToken sp:IncludeToken="xs:anyURI" ? ... >  
3625 <wsp:Policy>  
3626 <fed:RequireReferenceToken11 ...>...</fed:RequireReferenceToken11> ?  
3627 ...  
3628 </wsp:Policy> ?  
3629 ...  
3630 </fed:RequireReferenceToken>
```

3631 The following describes the attributes and elements listed in the schema outlined above:

3632 /fed:RequireReferenceToken

3633 This identifies a RequireReference assertion

3634 /fed:RequireReferenceToken/sp:IncludeToken

3635 This OPTIONAL attribute identifies the token inclusion value for this token assertion

3636 /fed:RequireReferenceToken/wsp:Policy

3637 This OPTIONAL element identifies additional requirements for use of the
3638 fed:RequireReferenceToken assertion.

3639 /fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11

3640 This OPTIONAL element indicates that a reference token should be used as defined in this
3641 specification.

3642 /fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11/@{any}

3643 This extensibility mechanism allows attributes to be added. Use of this extensibility point MUST
3644 NOT violate or alter the semantics defined in this specification.

3645 /fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11/{any}

3646 This is an extensibility point allowing content elements to be specified. Use of this extensibility
3647 point MUST NOT alter semantic defined in this specification.

3648 /fed:RequireReferenceToken/@{any}

3649 This extensibility mechanism allows attributes to be added . Use of this extensibility point MUST
3650 NOT violate or alter the semantics defined in this specification.

3651 /fed:RequireReferenceToken/{any}

3652 This is an extensibility point allowing content elements to be specified. Use of this extensibility
3653 point MUST NOT alter semantic defined in this specification.

3654 This assertion is used wherever acceptable token types are identified (e.g. within the supporting token
3655 assertions defined in WS-SecurityPolicy).

3656 14.2 WebBinding Assertion

3657 The WebBinding assertion is used in scenarios where requests are made of token services using a Web
3658 client and HTTP with GET, POST, and redirection as described in Section 13. Specifically, this assertion
3659 indicates that the requests use the Web client mechanism defined in this document and are protected
3660 using the means provided by a transport. This binding has several specific binding properties:

- 3661 • The [TransportToken] property indicates what transport mechanism is used to protect requests
3662 and responses.
- 3663 • The [AuthenticationToken] property indicates the REQUIRED token type for authentication. Note
3664 that this can be a choice of formats as it uses nested policy. Also note that this can specify
3665 fed:ReferenceToken as an option to indicate that token handles are accepted (and dereferenced).
- 3666 • The [RequireSignedTokens] property indicates that tokens MUST be signed i.e. only tokens that
3667 are signed are accepted.
- 3668 • The [RequireBearerTokens] property indicates that tokens MUST be bearer tokens i.e only
3669 bearer tokens are accepted.
- 3670 • The [RequireSharedCookies] property indicates if shared cookies MUST be used for home realm
3671 discovery
- 3672 • The [Base64Url] property indicates that base64url encoded xml parameter content is REQUIRED.

3673 The syntax is as follows:

```
3674 <fed:WebBinding ...>  
3675   <wsp:Policy>  
3676     <sp:TransportToken ...> ... </sp:TransportToken> ?  
3677     <fed:AuthenticationToken ... > ?  
3678       <wsp:Policy> ... </wsp:Policy>  
3679       <fed:ReferenceToken ...>... </fed:ReferenceToken> ?  
3680     </fed:AuthenticationToken>   <fed:RequireSignedTokens ... /> ?  
3681     <fed:RequireBearerTokens ... /> ?  
3682     <fed:RequireSharedCookies ... /> ?  
3683     <fed:Base64Url ... /> ?  
3684     ...  
3685   </wsp:Policy> ?  
3686 </fed:WebBinding>
```

3687 The following describes the attributes and elements listed in the schema outlined above:

3688 /fed:WebBinding

3689 This identifies a WebBinding assertion

3690 /fed:WebBinding/wsp:Policy

3691 This identifies a nested `wsp:Policy` element that defines the behavior of the WebBinding
3692 assertion.

3693 /fed:WebBinding/wsp:Policy/sp:TransportToken

3694 This indicates that a Transport Token as defined in [WS-SecurityPolicy] is REQUIRED

3695 /fed:WebBinding/wsp:Policy/fed:AuthenticationToken

3696 This indicates the REQUIRED token type for authentication.

3697 /fed:WebBinding/wsp:Policy/fed:AuthenticationToken/wsp:Policy

3698 This indicates a nested `wsp:Policy` element to specify a choice of formats for the authentication
3699 token.

3700 /fed:WebBinding/wsp:Policy/fed:AuthenticationToken/fed:ReferenceToken

- 3701 This OPTIONAL element indicates token handles that are accepted. See section 8.1 for a
3702 complete description.
- 3703 /fed:WebBinding/wsp:Policy/RequireSignedTokens
- 3704 This indicates a requirement for tokens to be signed. This sets the [RequireSignedTokens]
3705 property to true (the default value is false).
- 3706 /fed:WebBinding/wsp:Policy/RequireBearerTokens
- 3707 This indicates a requirement for bearer tokens. This sets the [RequireBearerTokens] property to
3708 true (the default value is false).
- 3709 /fed:WebBinding/wsp:Policy/RequireSharedCookies
- 3710 This indicates a requirement for shared cookies to facilitate home realm discovery. This sets the
3711 [RequireSharedCookies] property to true (the default value is false).
- 3712 /fed:WebBinding/wsp:Policy/Base64Url
- 3713 This indicates a requirement for xml parameter content to be base64url encoded. This sets the
3714 [Bas64Url] property to true (the default value is false).
- 3715 Note that the `sp:AlgorithmSuite`, `sp:Layout`, and `sp:IncludeTimestamp` properties are not used
3716 by this binding and SHOULD NOT be specified.
- 3717 This assertion SHOULD only be used with endpoint subjects.

3718 14.3 Authorization Policy

- 3719 To indicate support for the authorization features described in this specification, the following policy
3720 assertions are specified.

```
3721 <fed:RequiresGenericClaimDialect ... />
3722 <fed:IssuesSpecificMetadataFault ... />
3723 <fed:AdditionalContextProcessed ... />
```

- 3724 The following describes the above syntax:

3725 /fed:RequiresGenericClaimDialect

- 3726 This assertion indicates that the use of the generic claim dialect defined in this specification in
3727 Section 9.3 is REQUIRED by the service.

3728 /fed:IssuesSpecificPolicyFault

- 3729 This assertion indicates that the service issues the `fed:SpecificPolicy` Fault defined in this
3730 document if the security requirements for a specific request are beyond those of the base policy.

3731 /fed:AdditionalContextProcessed

- 3732 This assertion indicates that the service will process the `fed:AdditionalContext` parameter if
3733 specified in an RST request.

3734 Typically these assertions are specified at the service or port/endpoint.

3735 These assertions SHOULD be specified within a binding assertion.

3736

15 Error Handling

3737

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

3738

These errors use the SOAP Fault mechanism. Note that the reason text provided below is

3739

RECOMMENDED, but alternative text MAY be provided if more descriptive or preferred by the

3740

implementation. The table below is defined in terms of SOAP 1.1. For SOAP 1.2 the Fault/Code/Value is

3741

env:Sender (as defined in SOAP 1.2) and the Fault/Code/SubCode/Value is the *faultcode* below, and the

3742

Fault/Reason/Text is the *faultstring* below. It should be noted that profiles MAY provide second-level

3743

detail fields but they should be careful not to introduce security vulnerabilities when doing so (e.g. by

3744

providing too detailed information or echoing confidential information over insecure channels). It is

3745

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://docs.oasis-open.org/wsfed/federation/200706/Fault/NoPseudonymInScope
The principal is already signed in (need not be reported)	fed:AlreadySignedIn	http://docs.oasis-open.org/wsfed/federation/200706/Fault/AlreadySignedIn
The principal is not signed in (need not be reported)	fed:NotSignedIn	http://docs.oasis-open.org/wsfed/federation/200706/Fault/NotSignedIn
An improper request was made (e.g., Invalid/unauthorized pseudonym request)	fed:BadRequest	http://docs.oasis-open.org/wsfed/federation/200706/Fault/BadRequest
No match for the specified scope	fed:NoMatchInScope	http://docs.oasis-open.org/wsfed/federation/200706/Fault/NoMatchInScope
Credentials provided don't meet the freshness requirements	fed:NeedFresherCredentials	http://docs.oasis-open.org/wsfed/federation/200706/Fault/NeedFresherCredentials
Specific policy applies to the request – the new policy is specified in the S12:Detail element.	fed:SpecificPolicy	http://docs.oasis-open.org/wsfed/federation/200706/Fault/SpecificPolicy

Error that occurred (faultstring)	Fault code (faultcode)	Fault Action URI
The specified dialect for claims is not supported	fed:UnsupportedClaimsDialect	http://docs.oasis-open.org/wsfed/federation/200706/Fault/UnsupportedClaimsDialect
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://docs.oasis-open.org/wsfed/federation/200706/Fault/RstParameterNotAccepted
A desired issuer name is not supported by the STS	fed:IssuerNameNotSupported	http://docs.oasis-open.org/wsfed/federation/200706/Fault/IssuerNameNotSupported
A wencoding value or other parameter with XML content was received in an unknown/unsupported encoding.	fed:UnsupportedEncoding	http://docs.oasis-open.org/wsfed/federation/200706/Fault/UnsupportedEncoding

16 Security Considerations

3746

3747 It is strongly RECOMMENDED that the communication between services be secured using the
3748 mechanisms described in [WS-Security]. In order to properly secure messages, the body and all relevant
3749 headers need to be included in the signature.

3750 Metadata that is exchanged also needs to be secured to prevent various attacks. All metadata
3751 documents SHOULD be verified to ensure that the issuer can speak for the specified endpoint and that
3752 the metadata is what the issuer intended.

3753 All federation-related messages such as sign-out, principal, attribute, and pseudonym management
3754 SHOULD be integrity protected (signed or use transport security). If a message is received where the
3755 body is not integrity protected, it is RECOMMENDED that the message not be processed.

3756 All sign-out requests SHOULD be signed by the principal being purported to be signing in or out, or by a
3757 principal that is authorized to be on behalf of the indicated principal.

3758 It is also RECOMMENDED that all messages be signed by the appropriate security token service. If a
3759 message is received that does not have a signature from a principal authorized to speak for the security
3760 token service, it is RECOMMENDED that the message not be processed.

3761 When using Web messages care should be taken around processing of the *wreply* parameter as its value
3762 could be spoofed. It is RECOMMENDED that implementations do explicit lookup and verification of URL,
3763 and that these values be passed with transport security.

3764 The attribute service maintains information that may be very sensitive. Significant care SHOULD be
3765 taken to ensure that a principal's privacy is taken into account first and foremost.

3766 The pseudonym service may contain passwords or other information used in proof-of-possession
3767 mechanisms. Extreme care needs to be taken with this data to ensure that it cannot be compromised. It
3768 is strongly RECOMMENDED that such information be encrypted over communications channels and in
3769 any physical storage.

3770 If a security token does not contain an embedded signature (or similar integrity mechanism to protect
3771 itself), it SHOULD be included in any message integrity mechanisms (e.g. included in the message
3772 signature).

3773 If privacy is a concern, the security tokens used to authenticate and authorize messages MAY be
3774 encrypted for the authorized recipient(s) using mechanisms in WS-Security.

3775 Care SHOULD be taken when processing and responding to requests from 3rd-parties to mitigate
3776 potential information disclosure attacks by way of faulting requests for specific claims.

3777 As a general rule tokens SHOULD NOT have lifetimes beyond the minimum of the basis credentials
3778 (security tokens). However, in some cases special arrangements may exist and issuers may provide
3779 longer lived tokens. Care SHOULD be taken in such cases not to introduce security vulnerabilities.

3780 The following list summarizes common classes of attacks that apply to this protocol and identifies the
3781 mechanism to prevent/mitigate the attacks. Note that wherever WS-Security is suggested as the
3782 mitigation, [HTTPS] is the corresponding mechanism for Web requestors:

- 3783 • **Metadata alteration** – Alteration is prevented by including signatures in metadata or using secure
3784 channels for metadata transfer.
- 3785 • **Message alteration** – Alteration is prevented by including signatures of the message information
3786 using [WS-Security].
- 3787 • **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using [WS-Security].
- 3788 • **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by comparing
3789 secured policies – see [WS-Policy] and [WS-SecurityPolicy]).

- 3790 • **Authentication** – Authentication is established using the mechanisms described in [WS-Security]
3791 and [WS-Trust]. Each message is authenticated using the mechanisms described in [WS-Security].
- 3792 • **Accountability** – Accountability is a function of the type of and string of the key and algorithms being
3793 used. In many cases, a strong symmetric key provides sufficient accountability. However, in some
3794 environments, strong PKI signatures are required.
- 3795 • **Availability** – All reliable messaging services are subject to a variety of availability attacks. Replay
3796 detection is a common attack and it is RECOMMENDED that this be addressed by the mechanisms
3797 described in [WS-Security]. Other attacks, such as network-level denial of service attacks are harder
3798 to avoid and are outside the scope of this specification. That said, care SHOULD be taken to ensure
3799 that minimal state is saved prior to any authenticating sequences.
- 3800 • **Replay attacks:** It is possible that requests for security tokens could be replayed. Consequently, it
3801 is RECOMMENDED that all communication between Security Token Services and resources take
3802 place over secure connections. All cookies indicating state SHOULD be set as secure.
- 3803 • **Forged security tokens:** Security token services MUST guard their signature keys to prevent
3804 forging of tokens and requestor identities.
- 3805 • **Privacy:** Security token services SHOULD NOT send requestors' personal identifying information or
3806 information without getting consent from the requestor. For example a Web site SHOULD NOT
3807 receive requestors' personal information without an appropriate consent process.
- 3808 • **Compromised services:** If a Security Token Service is compromised, all requestor accounts
3809 serviced SHOULD be assumed to be compromised as well (since an attacker can issue security
3810 tokens for any account they want). However they SHOULD NOT not be able to issue tokens directly
3811 for identities outside the compromised realm. This is of special concern in scenarios like the 3rd party
3812 brokered trust where a 3rd party IP/STS is brokering trust between two realms. In such a case
3813 compromising the broker results in the ability to indirectly issue tokens for another realm by indicating
3814 trust.
- 3815 As with all communications careful analysis SHOULD be performed on the messages and interactions to
3816 ensure they meet the desired security requirements.
3817

Appendix A WSDL

3818

3819 The following illustrates the WSDL for the Web service methods described in this specification:

```
3820 <wsdl:definitions xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
3821     xmlns:xs='http://www.w3.org/2001/XMLSchema'
3822     xmlns:tns='http://docs.oasis-open.org/wsfed/federation/200706'
3823     targetNamespace='http://docs.oasis-open.org/wsfed/federation/200706' >
3824
3825 <!-- WS-Federation endpoints implement WS-Trust -->
3826 <wsdl:import namespace='http://docs.oasis-open.org/ws-sx/ws-trust/200512
3827 location='http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3.wsdl'
3828 />
3829
3830 <!-- WS-Federation endpoints can implement WS-MEX -->
3831 <wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/09/mex'
3832 location='http://schemas.xmlsoap.org/ws/2004/09/mex/MetadataExchange.wsdl' />
3833
3834 <!-- WS-Federation endpoints can implement WS-Eventing -->
3835 <wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/08/eventing'
3836 location='http://schemas.xmlsoap.org/ws/2004/08/eventing/eventing.wsdl' />
3837
3838 <!-- WS-Federation endpoints can implement WS-Transfer -->
3839 <wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/09/transfer'
3840 location='http://schemas.xmlsoap.org/ws/2004/09/transfer/transfer.wsdl' />
3841
3842 <!-- WS-Federation endpoints can implement WS-ResourceTransfer -->
3843 <wsdl:import
3844 namespace='http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer'
3845 location='http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/wsrt.wsdl' />
3846
3847 <wsdl:types>
3848 <xs:schema
3849 <xs:import namespace='http://docs.oasis-open.org/wsfed/federation/200706' />
3850 </xs:schema>
3851 </wsdl:types>
3852
3853 <wsdl:message name='SignOut' >
3854 <wsdl:part name='Body' element='tns:SignOut' />
3855 </wsdl:message>
3856
3857 <wsdl:portType name='SignOutIn' >
3858 <wsdl:operation name='SignOut' >
3859 <wsdl:input message='tns:SignOut' />
3860 </wsdl:operation>
3861 </wsdl:portType>
3862
3863 <wsdl:portType name='SignOutOut' >
3864 <wsdl:operation name='SignOut' >
3865 <wsdl:output message='tns:SignOut' />
3866 </wsdl:operation>
3867 </wsdl:portType>
3868
3869 </wsdl:definitions>
```

3870

Appendix B Sample HTTP Flows for Web Requestor Detailed Example

3871

3872 This appendix provides sample HTTP messages for the detailed example previously described in the
3873 Web requestor section.

3874 In this example, the following URLs are used:

<i>Item</i>	<i>URL</i>
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

3875 Step 1 – GET resource

3876 GET https://res.resource.com/sales HTTP/1.1

3877 Step 2 – Redirect to resource's IP/STS

3878 HTTP/1.1 302 Found ↓
3879 Location:
3880 https://sts.resource.com/sts?wa=wsignin1.0&wreply=https://res.resource.com/sal
3881 es&wct=2003-03-03T19:06:21Z

3882 In addition, the resource could check for a previously written artifact/cookie and, if present, skip to Step
3883 10.

3884 Step 3 – GET resource challenge

3885 GET https://sts.resource.com/sts?wa=wsignin1.0&wreply=
3886 https://res.resource.com/sales&wct=2003-03-03T19:06:21Z HTTP/1.1

3887 Step 3.1 – UI to determine realm (OPTIONAL)

3888 [Implementation Specific Traffic]

3889 Step 4 – Redirect to requestor's IP/STS

3890 HTTP/1.1 302 Found ↓
3891 Location: https://sts.account.com/sts?wa=wsignin1.0&wreply=
3892 https://sts.resource.com/sts&wctx= https://res.resource.com/sales&wct=2003-03-
3893 03T19:06:22Z&wtrealm=resource.com

3894 In addition, the Resource IP/STS MAY check for a previously written artifact/cookie and, if present, skip to
3895 Step 8.

3896 Step 5 – Requestor IP/STS challenge

3897 GET
3898 https://sts.account.com/sts?wa=wsignin1.0&wreply=https://sts.resource.com/sts&
3899 wctx=https://res.resource.com/sales&wct=2003-03-
3900 03T19:06:22Z&wtrealm=resource.com HTTP/1.1

3901 Step 5.1 – UI to collect authentication data (OPTIONAL)

3902

[Implementation Specific Traffic]

3903

Step 6 – Return requestor token

3904

```
HTTP/1.1 200 OK
```

3905

```
...
```

3906

3907

```
<html xmlns="https://www.w3.org/1999/xhtml">
```

3908

```
<head>
```

3909

```
<title>Working...</title>
```

3910

```
</head>
```

3911

```
<body>
```

3912

```
<form method="post" action="https://sts.resource.com/sts">
```

3913

```
<p>
```

3914

```
<input type="hidden" name="wa" value="wsignin1.0" />
```

3915

```
<input type="hidden" name="wctx" value="https://res.resource.com/sales" />
```

3916

```
<input type="hidden" name="wresult"
```

3917

```
value="&lt;RequestSecurityTokenResponse&gt;...&lt;/RequestSecurityTokenResponse
```

3918

```
e&gt;" />
```

3919

```
<button type="submit">POST</button> <!-- included for requestors that do not
```

3920

```
support javascript -->
```

3921

```
</p>
```

3922

```
</form>
```

3923

```
<script type="text/javascript">
```

3924

```
setTimeout('document.forms[0].submit()', 0);
```

3925

```
</script>
```

3926

```
</body>
```

3927

```
</html>
```

3928

Step 7 – POST requestor token

3929

```
POST https://sts.resource.com/sts HTTP/1.1 ↵
```

3930

```
... ↵
```

3931

```
↵
```

3932

```
wa=wsignin1.0 ↵
```

3933

```
wctx=https://res.resource.com/sales
```

3934

```
wresult=<RequestSecurityTokenResponse>...</RequestSecurityTokenResponse>
```

3935

Step 8 – Return resource token

3936

```
HTTP/1.1 200 OK
```

3937

```
...
```

3938

3939

```
<html xmlns="https://www.w3.org/1999/xhtml">
```

3940

```
<head>
```

3941

```
<title>Working...</title>
```

3942

```
</head>
```

3943

```
<body>
```

3944

```
<form method="post" action="https://res.resource.com/sales">
```

3945

```
<p>
```

3946

```
<input type="hidden" name="wa" value="wsignin1.0" />
```

3947

```
<input type="hidden" name="wresult"
```

3948

```
value="&lt;RequestSecurityTokenResponse&gt;...&lt;/RequestSecurityTokenResponse
```

3949

```
e&gt;" />
```

3950

```
<button type="submit">POST</button> <!-- included for requestors that do not
```

3951

```
support javascript -->
```

3952

```
</p>
```

3953

```
</form>
```

3954

```
<script type="text/javascript">
```

3955

```
setTimeout('document.forms[0].submit()', 0);
```

3956

```
</script>
```

3957

```
</body>
```

3958

```
</html>
```

3959 **Step 9 – POST Resource token**

```
3960 POST https://res.resource.com/sales HTTP/1.1 ↵  
3961 ... ↵  
3962 ↵  
3963 wa=wsignin1.0 ↵  
3964 wresult=<RequestSecurityTokenResponse>...</RequestSecurityTokenResponse>
```

3965 **Step 10 – Return result**

```
3966 [Implementation Specific Traffic]
```

3967

Appendix C Sample Use Cases

3968
3969
3970
3971

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.

3972

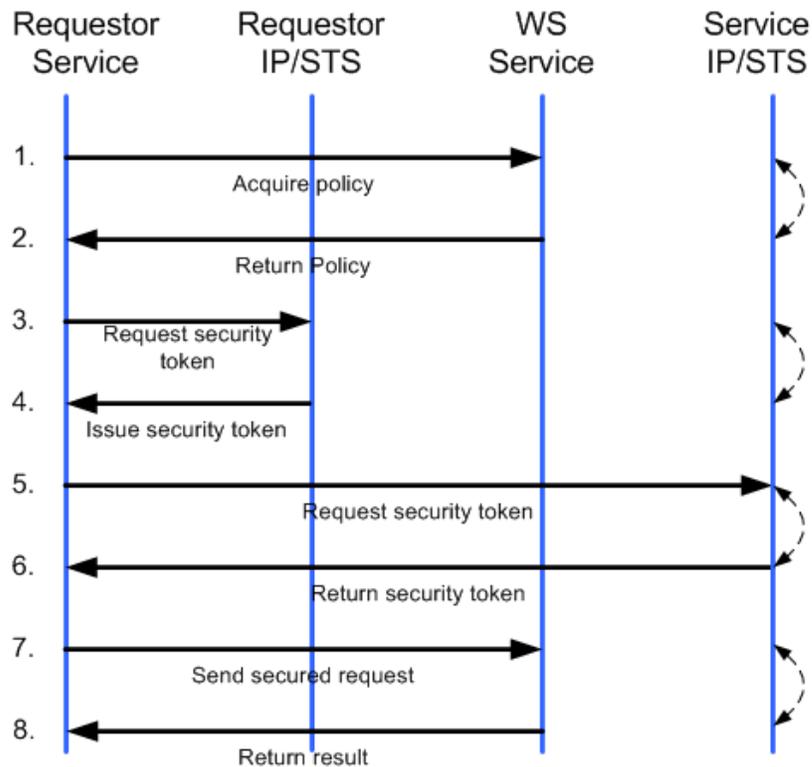
C.1 Single Sign On

3973
3974

Requestors use the mechanisms defined within [WS-Security], [WS-Trust], and [WS-Federation] to effect single sign-on.

3975
3976
3977
3978
3979
3980
3981
3982

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.



3983

3984
3985

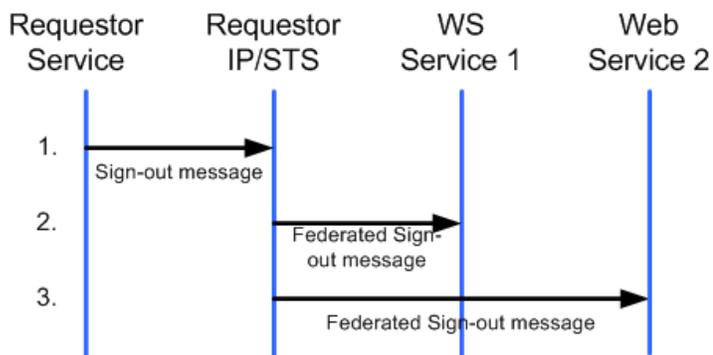
While the example above doesn't illustrate this, it is possible that the WS-Trust messages for security tokens MAY involve challenges to the requestors. Refer to WS-Trust for additional information.

3986 **C.2 Sign-Out**

3987 Just as it isn't typical for Web Service requestors to sign-in as a special operation, it isn't typical to *sign-*
3988 *out* either. However, for those scenarios where this is desirable, the sign-out messages defined in this
3989 specification can be used.

3990 In situations where federated sign-out messages are desirable, the requestor's IP/STS SHOULD keep
3991 track of the realms to which it has issued tokens – specifically the IP/STS for the realms (or resources if
3992 different). When the sign-out is received at the requestor's IP/STS, the requestor's IP/STS is responsible
3993 for issuing federated sign-out messages to interested and authorized parties. The exact mechanism by
3994 which this occurs is up to the IP/STS, but it is strongly RECOMMENDED that the sign-out messages
3995 defined in WS-Federation be used.

3996 When a federated sign-out message is received at a realm, the realm SHOULD clean-up any cached
3997 information and delete any associated state as illustrated in the figure below:

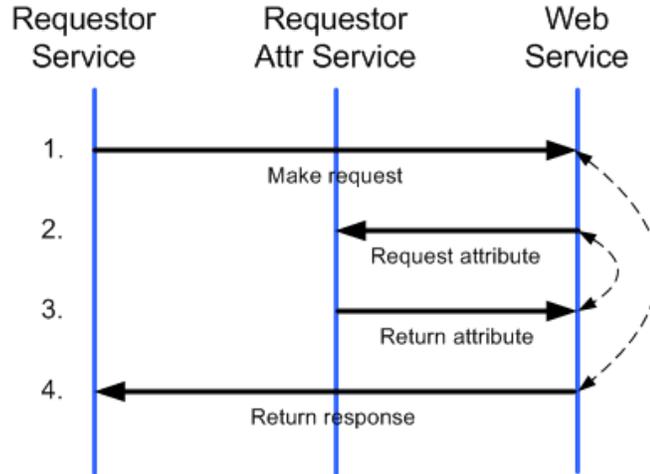


3998

3999 **C.3 Attributes**

4000 For Web Service requestors, attribute services are identified via WS-Policy or metadata as previously
4001 described. Web services and other authorized parties can obtain or even update attributes using the
4002 messages defined by the specific attribute service.

4003 The figure below illustrates a scenario where a requestor issues a request to a Web service. The request
4004 MAY include the requestor's policy or it may MAY be already cached at the service or the requestor MAY
4005 use [WS-MetadataExchange]. The Web service issues a request to the requestor's attribute service to
4006 obtain the values of a few attributes; WS-Policy MAY be used to describe the location of the attribute
4007 service. The service is authorized so the attributes are returned. The request is processed and a
4008 response is returned to the requestor.

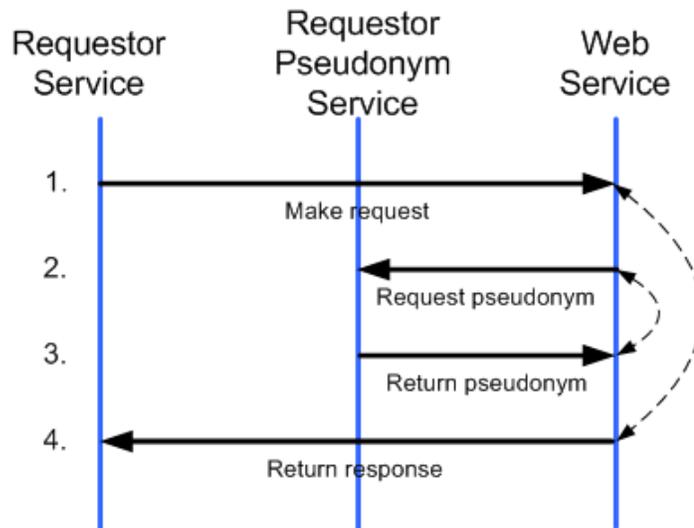


4009

4010 C.4 Pseudonyms

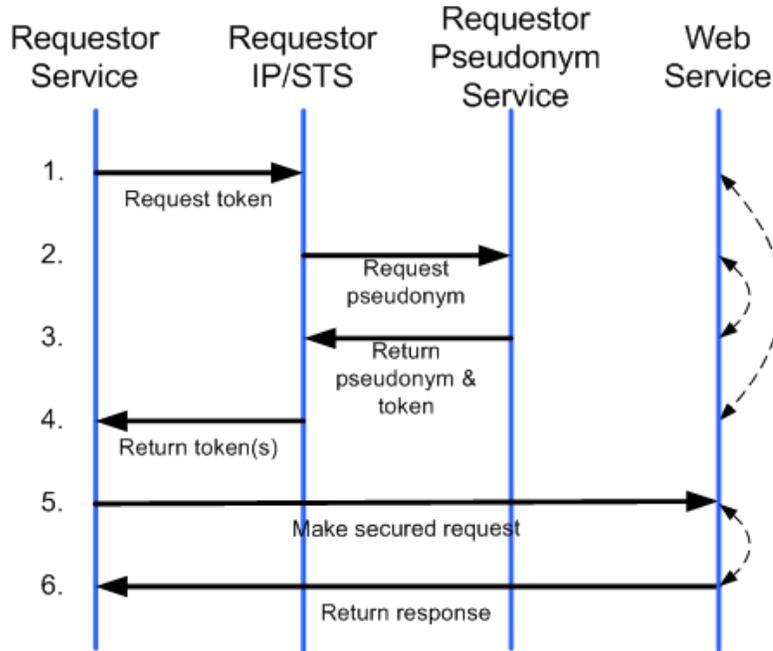
4011 For Web Service requestors, pseudonym services are identified via metadata as previously described.
 4012 Services and other authorized parties can obtain or manage pseudonyms using the messages previously
 4013 defined.

4014 The figure below illustrates a scenario where a requestor issues a request to a Web service. The request
 4015 MAY include the requestor's policy and the location of the requestor's pseudonym service or it MAY be
 4016 already cached at the Web service. The Web service issues a request to the requestor's pseudonyms
 4017 service to obtain the pseudonyms that are authorized by the security token. The Web service is
 4018 authorized so the pseudonym is returned. The request is processed and a response is returned to the
 4019 requestor.



4020

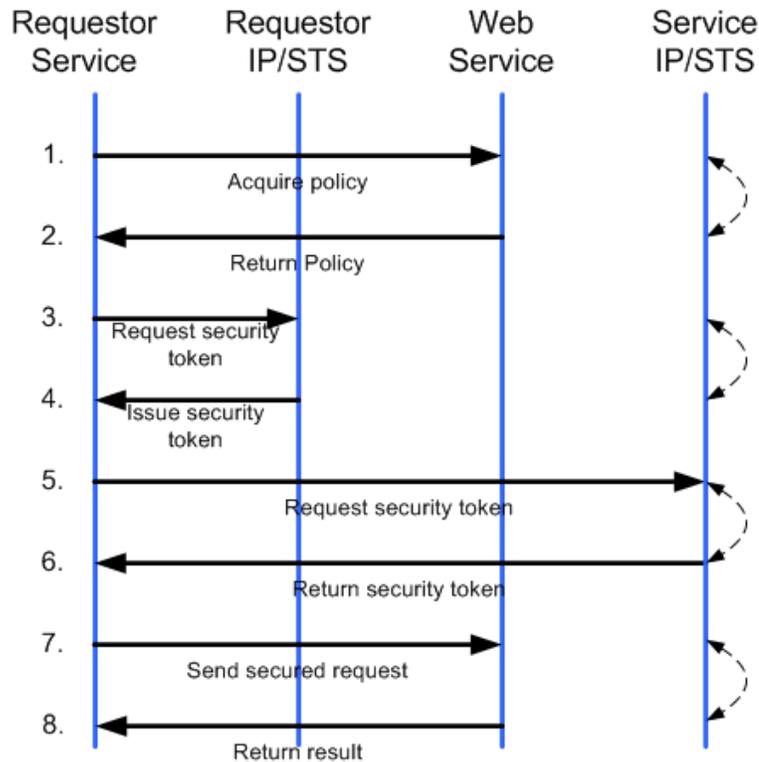
4021 As previously described, the pseudonym and IP/STS can interact as part of the token issuance process.
 4022 The figure below illustrates a scenario where a requestor has previously associated a pseudonym and a
 4023 security token for a specific realm. When the requestor requests a security token to the domain/realm,
 4024 the pseudonym and token are obtained and returned to the requestor. The requestor uses these security
 4025 tokens for accessing the Web service.



4026

4027 **C.5 Detailed Example**

4028 This section provides a detailed example of the protocol defined in this specification. The exact flow can
 4029 vary significantly; however, the following diagram and description depict a *common* sequence of events.
 4030 In this scenario, a SOAP requestor is attempting to access a service which requires security
 4031 authentication to be validated by the resource's security token service.

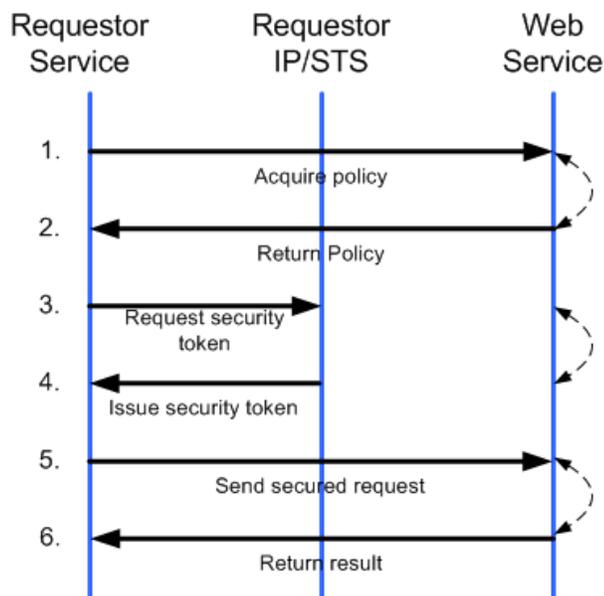


4032

4033 **Step 1: Acquire Policy**
 4034 If the requestor doesn't already have the policy for the service, it can obtain it using the mechanisms
 4035 defined in WS-MetadataExchange.
 4036 **Step 2: Return Policy**
 4037 The requested policy is returned using the mechanisms defined in WS-MetadataExchange.
 4038 **Step 3: Request Security Token**
 4039 The requestor requests a security token from its IP/STS (assuming short-lived security tokens) using the
 4040 mechanisms defined in WS-Trust (<RequestSecurityToken>)
 4041 **Step 4: Issue Security Token**
 4042 The IP/STS returns a security token (and optional proof of possession information) using the mechanisms
 4043 defined in WS-Trust (<RequestSecurityTokenResponse> and <RequestedProofToken>)
 4044 **Step 5: Request Security Token**
 4045 The requestor requests a security token from the Web services IP/STS for the target Web service using
 4046 the mechanisms defined in WS-Trust (<RequestSecurityToken>). Note that this is determined via
 4047 policy or some out-of-band mechanism.
 4048 **Step 6: Issue Security Token**
 4049 The Web service's IP/STS returns a token (and optionally proof of possession information) using the
 4050 mechanisms defined in WS-Trust (<RequestSecurityTokenResponse>)
 4051 **Step 7: Send secured request**
 4052 The requestor sends the request to the service attaching and securing the message using the issued
 4053 tokens as described in WS-Security.
 4054 **Step 8: Return result**
 4055 The service issues a secured reply using its security token.

4056 C.6 No Resource STS

4057 The figure below illustrates the resource access scenario above, but without a resource STS. That is, the
 4058 Web service acts as its own STS:



4059

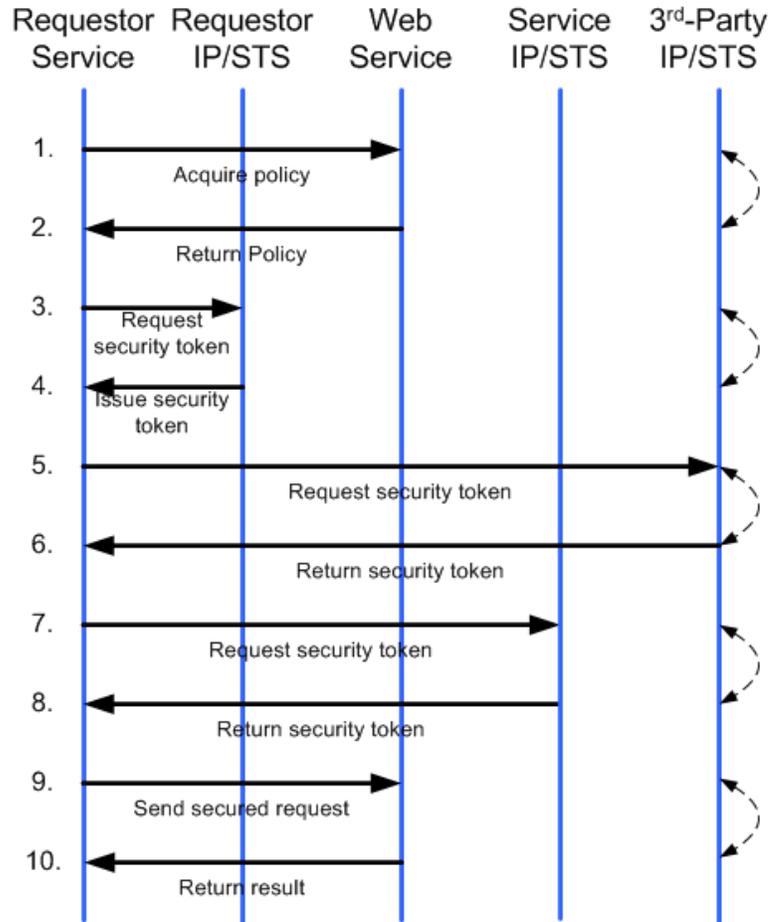
4060

C.7 3rd-Party STS

4061

The figure below illustrates the resource access scenario above, but trust is brokered through a 3rd-party STS:

4062



4063

4064

Note that 3rd-Party IP/STS is determined via policy or some out-of-band mechanism.

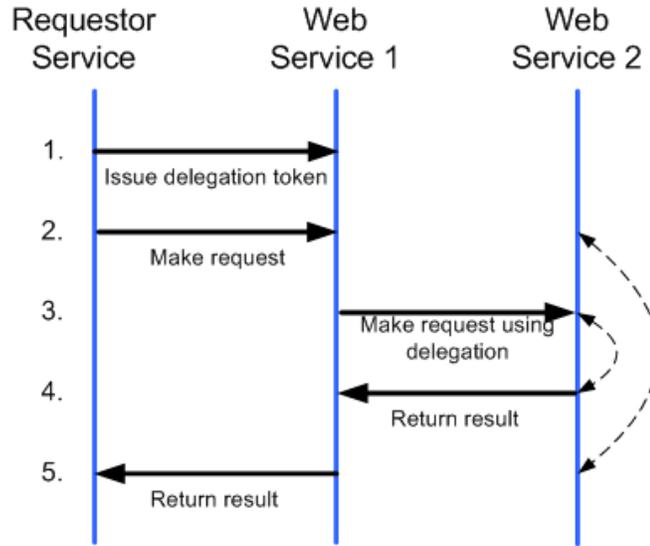
4065

C.8 Delegated Resource Access

4066

The figure below illustrates where a resource accesses data from another resource on behalf of the requestor:

4067



4068

4069 In this example, the requestor used a `<RequestSecurityTokenResponse>` as defined in WS-Trust to
 4070 issue the delegation token in Step 1. This provides to Web Service 1 the necessary information so that
 4071 Web Service 1 can act on the requestor's behalf as it contacts Web Service 2.

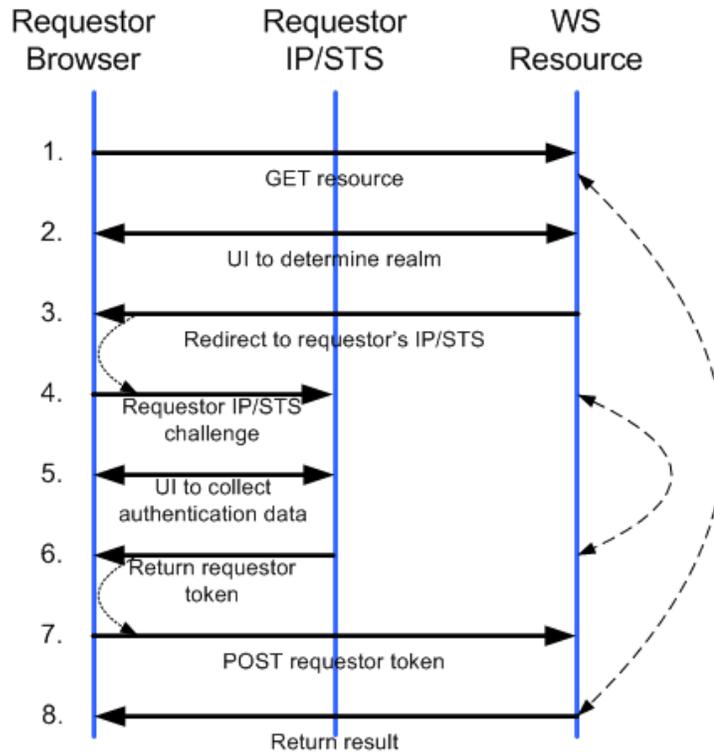
4072

4073 **C.9 Additional Web Examples**

4074 This section presents interaction diagrams for additional Web requestor scenarios.

4075 **No Resource STS**

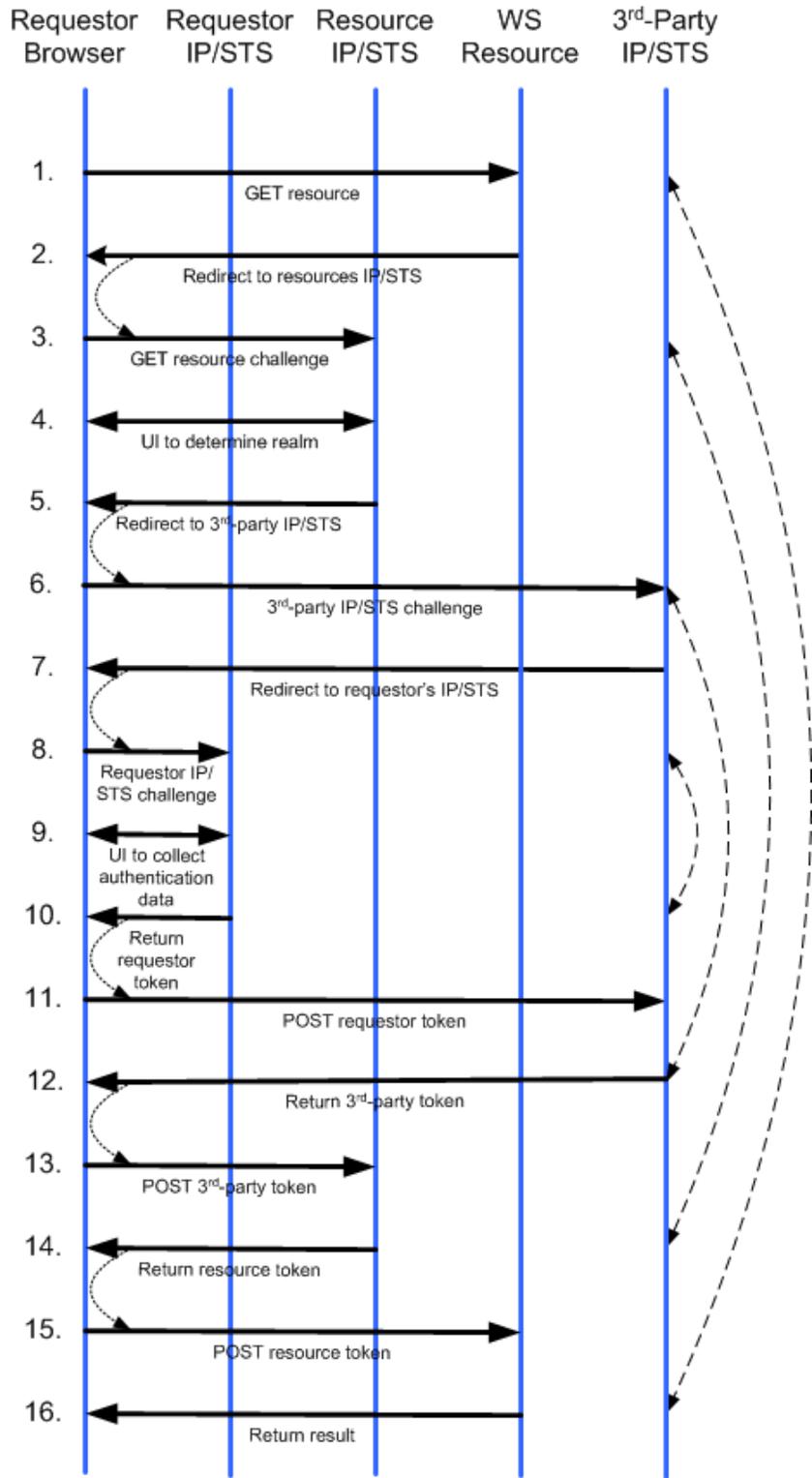
4076 The figure below illustrates the sign-in scenario above, but without a resource STS. That is, the requestor
 4077 acts as its own STS:



4078

4079 **3rd-Party STS**

4080 The figure below illustrates the sign-in scenario above, but trust is brokered through a 3rd-party STS:

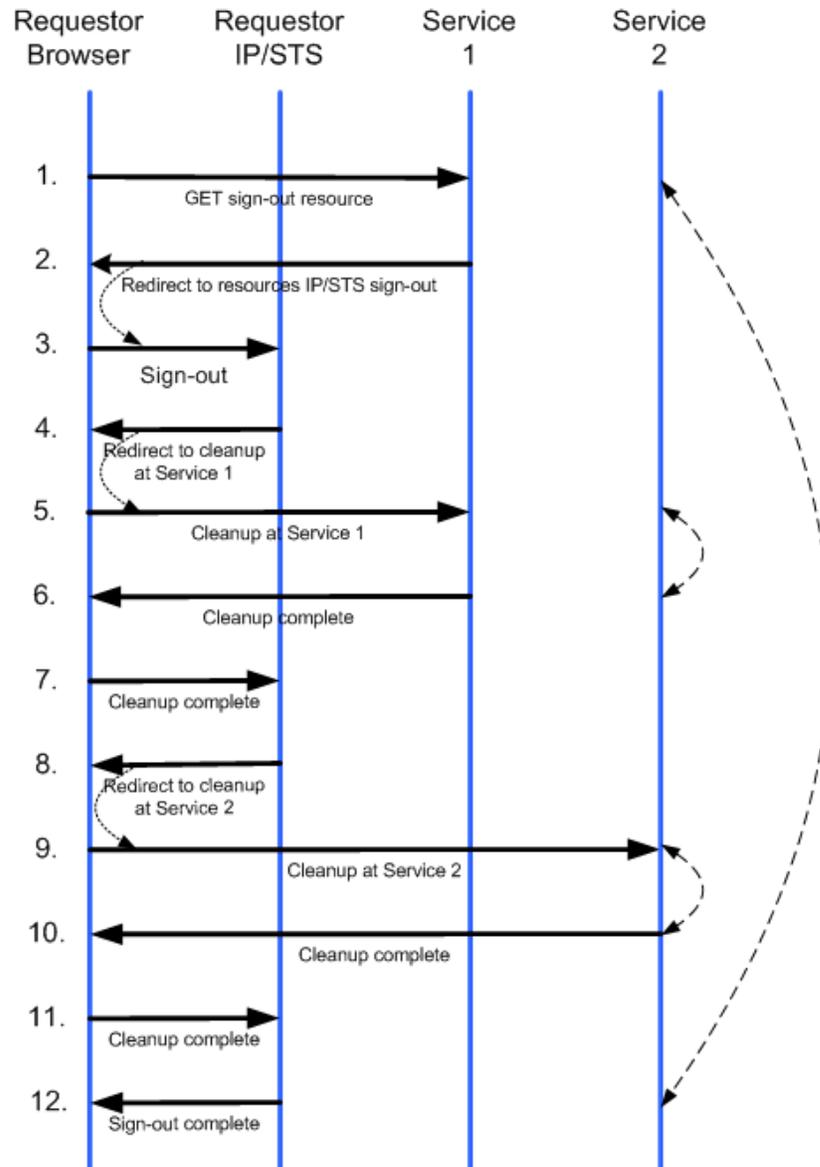


4081

4082 **Sign-Out**

4083 The figure below illustrates the sign-out flow for a Web browser requestor that has signed in at two sites
 4084 and requests that the sign-out cleanup requests redirect back to the requestor: The message flow is an

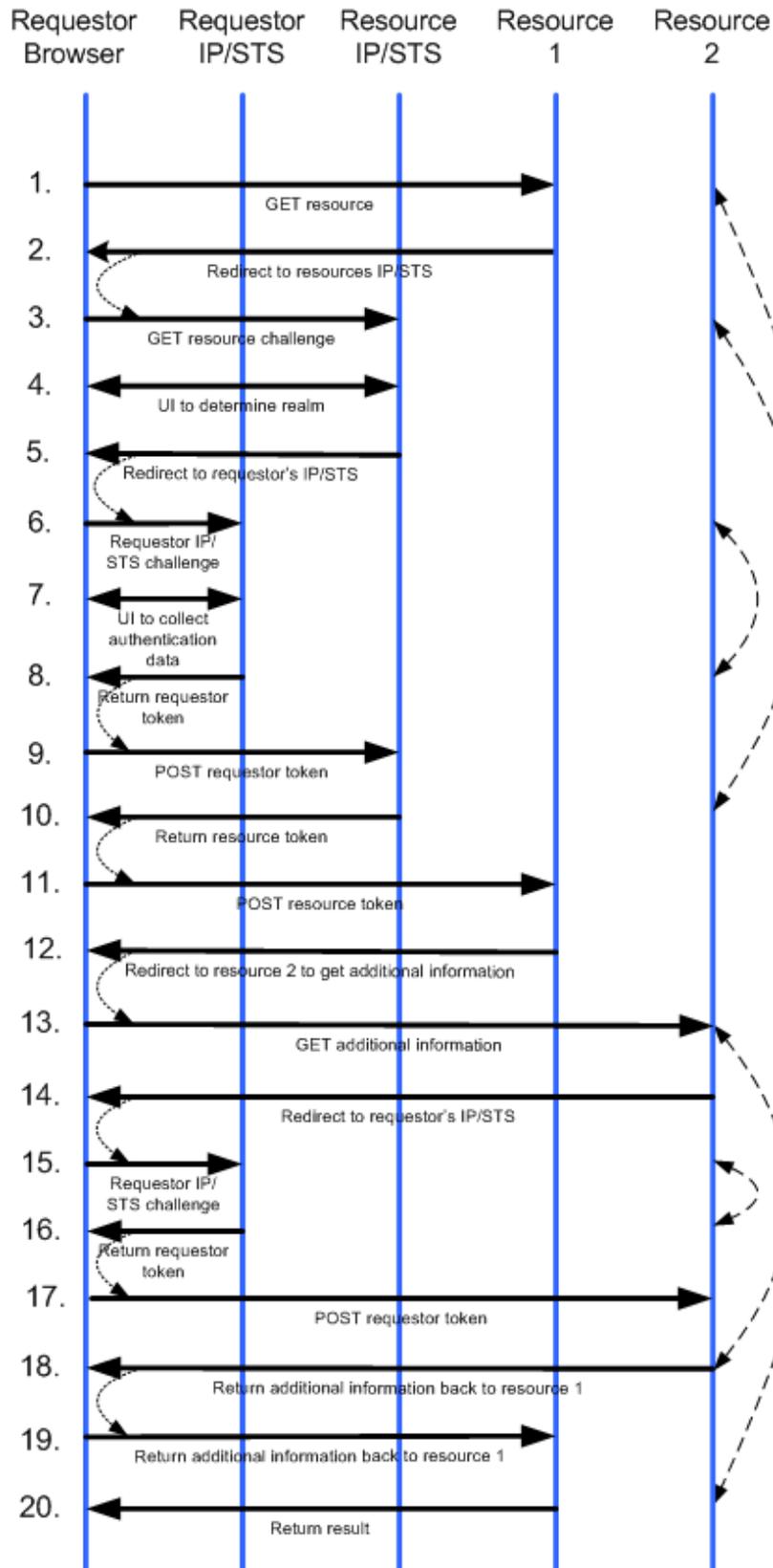
4085 example of the use case in which all sign-out messages must be transmitted by the requestor. Since it
 4086 cannot be assumed that all browser requestors can transmit parallel requests, the sequential method is
 4087 depicted. This message flow is enabled by the "wreply" parameter defined in section 13.2.4.



4088

4089 **Delegated Resource Access**

4090 The figure below illustrates the case where a resource accesses data from another resource on behalf of
 4091 the first resource and the information is returned through the requestor:



4093

Appendix D SAML Binding of Common Claims

4094

The content of the auth:Value, auth:EncryptedValue, auth:StructuredValue, and auth:ConstrainedValue elements, not including the root node, can be serialized into any token format that supports the content format. For SAML 1.1 and 2.0 this content SHOULD be serialized into the saml:AttributeValue element.

4095

4096

4097

The display information, such as auth:DisplayName, auth:Description and auth:DisplayValue is not intended for serialization into tokens.

4098

4099

4100

Appendix E Acknowledgements

4101

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

4102

4103

Original Authors of the initial contributions:

4104

Hal Lockhart, BEA

4105

Steve Anderson, BMC Software

4106

Jeff Bohren, BMC Software

4107

Yakov Sverdlov, CA Inc.

4108

Maryann Hondo, IBM

4109

Hiroshi Maruyama, IBM

4110

Anthony Nadalin (Editor), IBM

4111

Nataraj Nagaratnam, IBM

4112

Toufic Boubez, Layer 7 Technologies, Inc.

4113

K Scott Morrison, Layer 7 Technologies, Inc.

4114

Chris Kaler (Editor), Microsoft

4115

Arun Nanda, Microsoft

4116

Don Schmidt, Microsoft

4117

Doug Walters, Microsoft

4118

Hervey Wilson, Microsoft

4119

Lloyd Burch, Novell, Inc.

4120

Doug Earl, Novell, Inc.

4121

Siddharth Bajaj, VeriSign

4122

Hemma Prafullchandra, VeriSign

4123

4124

Original Acknowledgements of the initial contributions:

4125

John Favazza, CA

4126

Tim Hahn, IBM

4127

Andrew Hatley, IBM

4128

Heather Hinton, IBM

4129

Michael McIntosh, IBM

4130

Anthony Moran, IBM

4131

Birgit Pfitzmann, IBM

4132

Bruce Rich, IBM

4133

Shane Weeden, IBM

4134

Jan Alexander, Microsoft

4135

Greg Carpenter, Microsoft

4136

Paul Cotton, Microsoft

4137

Marc Goodner, Microsoft

4138

Martin Gudgin, Microsoft

4139

Savas Parastatidis, Microsoft

4140

4141

TC Members during the development of this specification:

4142

Don Adams, TIBCO Software Inc.

4143

Steve Anderson, BMC Software

4144

Siddharth Bajaj, VeriSign

4145

Abbie Barbir, Nortel

4146

Hanane Becha, Nortel

4147

Toufic Boubez, Layer 7 Technologies Inc.

4148

Norman Brickman, Mitre Corporation

4149

Geoff Bullen, Microsoft Corporation

4150 Lloyd Burch, Novell
4151 Brian Campbell, Ping Identity Corporation
4152 Greg Carpenter, Microsoft Corporation
4153 Steve Carter, Novell
4154 Marco Carugi, Nortel
4155 Paul Cotton, Microsoft Corporation
4156 Doug Davis, IBM
4157 Fred Dushin, IONA Technologies
4158 Doug Earl, Novell
4159 Colleen Evans, Microsoft Corporation
4160 Christopher Ferris, IBM
4161 Marc Goodner, Microsoft Corporation
4162 Tony Gullotta, SOA Software Inc.
4163 Maryann Hondo, IBM
4164 Mike Kaiser, IBM
4165 Chris Kaler, Microsoft Corporation
4166 Paul Knight, Nortel
4167 Heather Kreger, IBM
4168 Ramanathan Krishnamurthy, IONA Technologies
4169 Kelvin Lawrence, IBM
4170 Paul Lesov, Wells Fargo
4171 David Lin, IBM
4172 Jonathan Marsh, WSO2
4173 Robin Martherus, Ping Identity Corporation
4174 Monica Martin, Microsoft Corporation
4175 Michael McIntosh, IBM
4176 Nandana Mihindukulasooriya, WSO2
4177 Anthony Nadalin, IBM
4178 Arun Nanda, Microsoft Corporation
4179 Kimberly Pease, Active Endpoints, Inc.
4180 Larry Rogers, Lockheed Martin
4181 Anil Saldhana, Red Hat
4182 Richard Sand, Tripod Technology Group, Inc.
4183 Don Schmidt, Microsoft Corporation
4184 Sidd Shenoy, Microsoft Corporation
4185 Kent Spaulding, Tripod Technology Group, Inc.
4186 David Staggs, Veterans Health Administration
4187 Yakov Sverdlov, CA
4188 Gene Thurston, AmberPoint
4189 Atul Tulshibagwale, Hewlett-Packard
4190 Ron Williams, IBM
4191 Jason Woloz, Booz Allen Hamilton
4192 Gerry Woods, SOA Software Inc.
4193