



Web Services Federation Language (WS-Federation) Version 1.2

Committee Draft 03

February 17 2009

Specification URIs:

This Version:

<http://docs.oasis-open.org/wsfed/federation/v1.2/cd/ws-federation-1.2-spec-cd-03.doc>
(Authoritative)
<http://docs.oasis-open.org/wsfed/federation/v1.2/cd/ws-federation-1.2-spec-cd-03.pdf>
<http://docs.oasis-open.org/wsfed/federation/v1.2/cd/ws-federation-1.2-spec-cd-03.html>

Previous Version:

<http://docs.oasis-open.org/wsfed/federation/v1.2/cd/ws-federation-1.2-spec-cd-02.doc>
<http://docs.oasis-open.org/wsfed/federation/v1.2/cd/ws-federation-1.2-spec-cd-02.pdf>
<http://docs.oasis-open.org/wsfed/federation/v1.2/cd/ws-federation-1.2-spec-cd-02.html>

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® 2009. 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	7
1.1	Document Roadmap	7
1.2	Goals and Requirements.....	8
1.2.1	Requirements.....	8
1.2.2	Non-Goals.....	9
1.3	Notational Conventions	9
1.4	Namespaces	10
1.5	Schema and WSDL Files	11
1.6	Terminology.....	11
1.7	Normative References.....	13
1.8	Non-Normative References.....	16
2	Model	17
2.1	Federation Basics.....	17
2.2	Metadata Model.....	20
2.3	Security Model.....	23
2.4	Trust Topologies and Security Token Issuance.....	23
2.5	Identity Providers.....	27
2.6	Attributes and Pseudonyms	27
2.7	Attributes, Pseudonyms, and IP/STS Services	31
3	Federation Metadata	33
3.1	Federation Metadata Document	33
3.1.1	Referencing Other Metadata Documents.....	35
3.1.2	Role Descriptor Types.....	37
3.1.3	LogicalServiceNamesOffered Element	43
3.1.4	PseudonymServiceEndpoints Element	43
3.1.5	AttributeServiceEndpoints Element.....	44
3.1.6	SingleSignOutSubscriptionEndpoints Element	44
3.1.7	SingleSignOutNotificationEndpoints Element	45
3.1.8	TokenTypesOffered Element	45
3.1.9	ClaimTypesOffered Element	46
3.1.10	ClaimTypesRequested Element.....	47
3.1.11	ClaimDialectsOffered Element	48
3.1.12	AutomaticPseudonyms Element	48
3.1.13	PassiveRequestorEndpoints Element	49
3.1.14	TargetScopes Element.....	49
3.1.15	[Signature] Property	50
3.1.16	Example Federation Metadata Document.....	51
3.2	Acquiring the Federation Metadata Document	52
3.2.1	WSDL.....	52
3.2.2	The Federation Metadata Path	53
3.2.3	Retrieval Mechanisms.....	53
3.2.4	FederatedMetadataHandler Header.....	54

3.2.5	Metadata Exchange Dialect	55
3.2.6	Publishing Federation Metadata Location	55
3.2.7	Federation Metadata Acquisition Security	57
4	Sign-Out.....	58
4.1	Sign-Out Message.....	58
4.2	Federating Sign-Out Messages.....	60
5	Attribute Service.....	62
6	Pseudonym Service	64
6.1	Filtering Pseudonyms	65
6.2	Getting Pseudonyms	66
6.3	Setting Pseudonyms	68
6.4	Deleting Pseudonyms	69
6.5	Creating Pseudonyms	69
7	Security Tokens and Pseudonyms.....	71
7.1	RST and RSTR Extensions.....	72
7.2	Username and Passwords	72
7.3	Public Keys	73
7.4	Symmetric Keys	73
8	Additional WS-Trust Extensions.....	74
8.1	Reference Tokens.....	74
8.2	Indicating Federations	75
8.3	Obtaining Proof Tokens from Validation.....	75
8.4	Client-Based Pseudonyms	76
8.5	Indicating Freshness Requirements	77
9	Authorization	78
9.1	Authorization Model.....	78
9.2	Indicating Authorization Context.....	78
9.3	Common Claim Dialect.....	80
9.3.1	Expressing value constraints on claims	82
9.4	Claims Target.....	84
9.5	Authorization Requirements	85
10	Indicating Specific Policy/Metadata.....	87
11	Authentication Types	89
12	Privacy.....	90
12.1	Confidential Tokens.....	90
12.2	Parameter Confirmation	91
12.3	Privacy Statements	92
13	Web (Passive) Requestors	94
13.1	Approach.....	94
13.1.1	Sign-On.....	94
13.1.2	Sign-Out.....	95
13.1.3	Attributes.....	96
13.1.4	Pseudonyms	97
13.1.5	Artifacts/Cookies	98

13.1.6 Bearer Tokens and Token References	98
13.1.7 Freshness	98
13.2 HTTP Protocol Syntax	99
13.2.1 Parameters	99
13.2.2 Requesting Security Tokens	100
13.2.3 Returning Security Tokens	102
13.2.4 Sign-Out Request Syntax	103
13.2.5 Attribute Request Syntax	104
13.2.6 Pseudonym Request Syntax	105
13.3 Detailed Example of Web Requester Syntax	105
13.4 Request and Result References	109
13.5 Home Realm Discovery	112
13.5.1 Discovery Service	112
13.6 Minimum Requirements	112
13.6.1 Requesting Security Tokens	112
13.6.2 Returning Security Tokens	113
13.6.3 Details of the RequestSecurityTokenResponse element	113
13.6.4 Details of the Returned Security Token Signature	114
13.6.5 Request and Response References	114
14 Additional Policy Assertions	115
14.1 RequireReferenceToken Assertion	115
14.2 WebBinding Assertion	116
14.3 Authorization Policy	117
15 Error Handling	118
16 Security Considerations	120
17 Conformance	122
Appendix A WSDL	123
Appendix B Sample HTTP Flows for Web Requestor Detailed Example	124
Appendix C Sample Use Cases	127
C.1 Single Sign On	127
C.2 Sign-Out	128
C.3 Attributes	128
C.4 Pseudonyms	129
C.5 Detailed Example	130
C.6 No Resource STS	131
C.7 3 rd -Party STS	132
C.8 Delegated Resource Access	132
C.9 Additional Web Examples	133
No Resource STS	133
3 rd -Party STS	134
Sign-Out	135
Delegated Resource Access	136
Appendix D SAML Binding of Common Claims	138
Appendix E Acknowledgements	139

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
wssell	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
md	urn:oasis:names:tc:SAML:2.0:metadata

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/v1.2/federation.xsd  
142 http://docs.oasis-open.org/wsfed/authorization/v1.2/authorization.xsd  
143 http://docs.oasis-open.org/wsfed/privacy/v1.2/privacy.xsd
```

144 The WSDL for this specification can be located at:

```
145 http://docs.oasis-open.org/wsfed/federation/v1.2/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. Berners-Lee, RFC 2616, "Hypertext Transfer Protocol -- HTTP/1.1". June 1999.
267		
268		
269		http://www.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 Levels", RFC 2119, Harvard University, March 1997.
273		
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]	W3C Member Submission, Web Services Metadata Exchange (WS-MetadataExchange), 13 August 2008
291		
292		http://www.w3.org/Submission/2008/SUBM-WS-MetadataExchange-20080813/
293		
294	[WS-Policy]	W3C Member Submission "Web Services Policy 1.2 - Framework", 25 April 2006.
295		
296		http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/
297		W3C Recommendation "Web Services Policy 1.5 – Framework", 04 September 2007
298		
299		http://www.w3.org/TR/2007/REC-ws-policy-20070904/
300	[WS-PolicyAttachment]	W3C Member Submission "Web Services Policy 1.2 - Attachment", 25 April 2006.
301		
302		http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/
303		
304		W3C Recommendation "Web Services Policy 1.5 – Attachment", 04 September 2007
305		
306		http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904/
307	[WS-SecurityPolicy]	OASIS Standard, "WS-SecurityPolicy 1.2", July 2007
308		http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702
309	[WS-Security]	OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", March 2004.
310		
311		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
312		
313		OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", February 2006.
314		
315		http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
316		
317	[WSS:UsernameToken]	OASIS Standard, "Web Services Security: UsernameToken Profile", March 2004
318		
319		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
320		

321		OASIS Standard, "Web Services Security: UsernameToken Profile 1.1", February 2006
322		
323		http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf
324		
325	[WSS:X509Token]	OASIS Standard, "Web Services Security X.509 Certificate Token Profile", March 2004
326		
327		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
328		
329		OASIS Standard, "Web Services Security X.509 Certificate Token Profile", February 2006
330		
331		http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf
332		
333	[WSS:KerberosToken]	OASIS Standard, "Web Services Security Kerberos Token Profile 1.1", February 2006
334		
335		http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf
336		
337	[WSS:SAMLTokenProfile]	OASIS Standard, "Web Services Security: SAML Token Profile", December 2004
338		
339		http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf
340		OASIS Standard, "Web Services Security: SAML Token Profile 1.1", February 2006
341		
342		http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf
343		
344	[WS-ResourceTransfer]	W3C Member Submission, "Web Services Resource Transfer (WS-ResourceTransfer)", 12 August 2008
345		
346		http://www.w3.org/Submission/2008/SUBM-WSRT-20080812/
347	[WS-Transfer]	W3C Member Submission, "Web Services Transfer (WS-Transfer)", 27 September 2006
348		
349		http://www.w3.org/Submission/2006/SUBM-WS-Transfer-20060927/
350	[WS-Trust]	OASIS Standard, "WS-Trust 1.3", March 2007
351		http://docs.oasis-open.org/ws-sx/ws-trust/200512
352	[ISO8601]	ISO Standard 8601:2004(E), "Data elements and interchange formats – Information interchange - Representation of dates and times", Third edition, December 2004
353		
354		
355		http://isotc.iso.org/livelink/livelink/4021199/ISO_8601_2004_E.zip?func=doc.Fetch&nodeid=4021199
356		
357	[DNS-SRV-RR]	Gulbrandsen, et al, RFC 2782, "DNS SRV RR", February 2000.
358		http://www.ietf.org/rfc/rfc2782.txt
359	[XML-Schema1]	W3C Recommendation, "XML Schema Part 1: Structures Second Edition", 28 October 2004.
360		
361		http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/
362	[XML-Schema2]	W3C Recommendation, "XML Schema Part 2: Datatypes Second Edition", 28 October 2004.
363		
364		http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/
365	[XML-C14N]	W3C Recommendation, "Canonical XML Version 1.0", 15 March 2001

366 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

367 W3C Recommendation, "Canonical XML Version 1.1", 2 May 2008

368 <http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/>

369 [XML-Signature] W3C Recommendation, "XML-Signature Syntax and Processing", 12
370 February 2002

371 <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>

372 W3C Recommendation, "XML Signature Syntax and Processing
373 (Second Edition)", 10 June 2008<http://www.w3.org/TR/2008/REC-xmlsig-core-20080610/>

374

375 [WSDL 1.1] W3C Note, "Web Services Description Language (WSDL 1.1)," 15
376 March 2001.

377 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

378 [XPath] W3C Recommendation "XML Path Language (XPath) Version 1.0", 16
379 November 1999.

380 <http://www.w3.org/TR/1999/REC-xpath-19991116>

381 [RFC 4648] S. Josefsson, et. al, RFC 4648 "The Base16, Base32, and Base64
382 Data Encodings" October 2006

383 <http://www.ietf.org/rfc/rfc4648.txt>

384 [Samlv2Meta] Metadata for the OASIS Security Assertion Markup Language (SAML)
385 V2.0. OASIS SSTC, September 2004.

386 Document ID sstc-saml-metadata-2.0-cd-03.

387 <http://www.oasis-open.org/committees/security/>

388

389 **1.8 Non-Normative References**

390

391 2 Model

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

394 2.1 Federation Basics

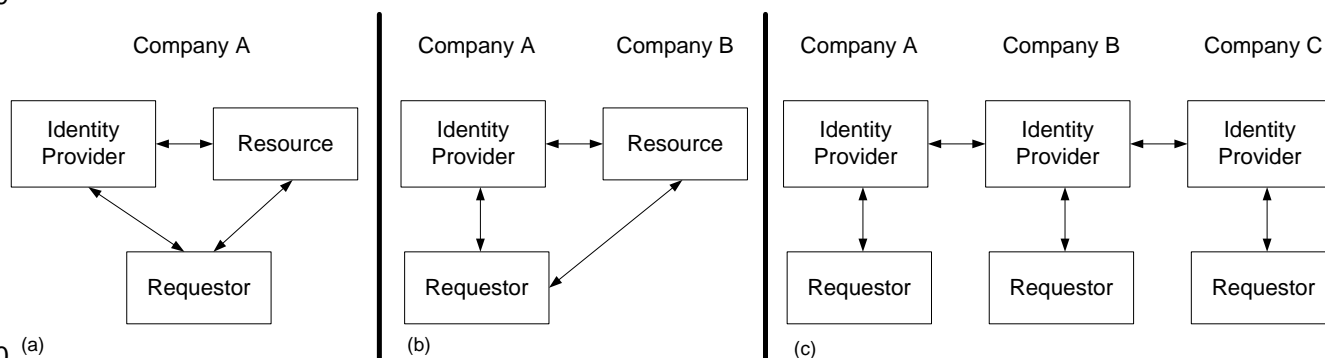
395 The goal of federation is to allow security principal identities and attributes to be shared across trust
396 boundaries according to established policies. The policies dictate, among other things, formats and
397 options, as well as trusts and privacy/sharing requirements.

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

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

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

409



410 (a)

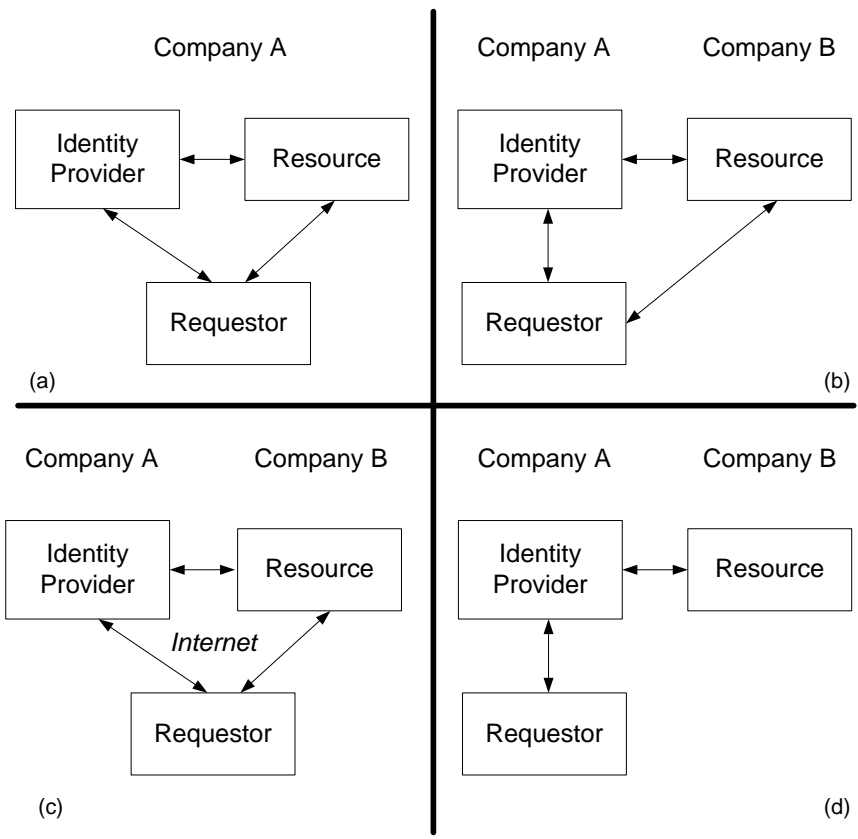
(b)

(c)

411

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

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

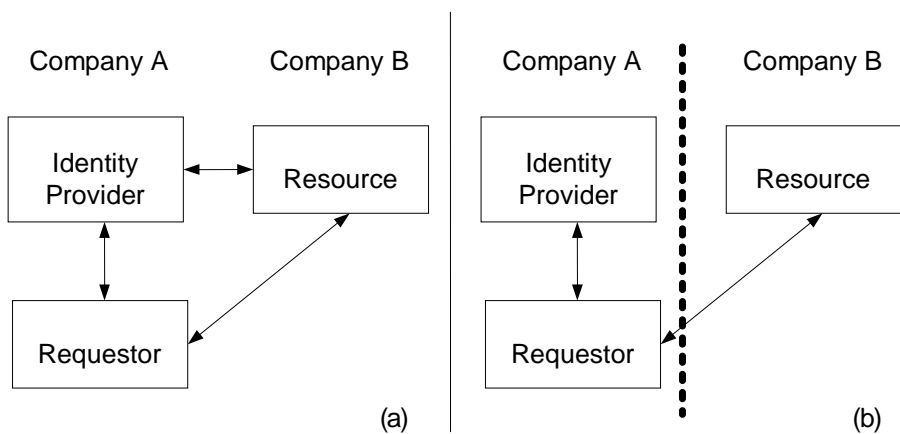


415

416

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

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



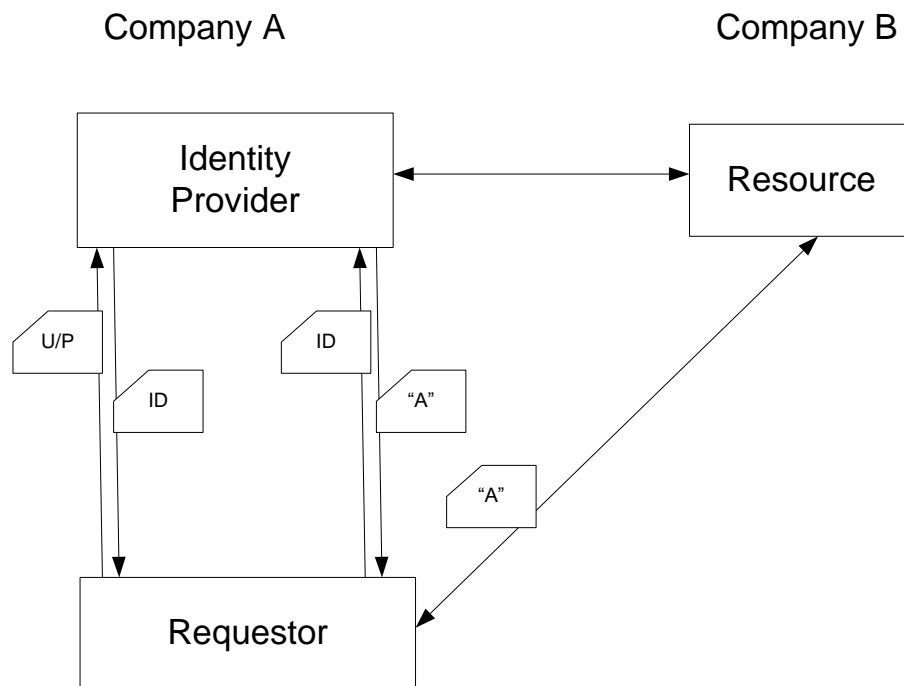
422

423

Figures 3a, 3b: Accessibility of Identity Provider

424 In the federation process some level of information is shared. The amount of information shared is
 425 governed by policy and often dictated by contract. This is because the information shared is often of a
 426 personal or confidential nature. For example, this may indicate name, personal identification numbers,

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



429

430

Figure 4: Sample Anonymous Access

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

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

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

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

455 the requestor's mapping service (called a pseudonym service) to map the given (potentially random)
456 digital identity into a constant service-specific digital identity which it has registered with the requestor's
457 mapping service. This also addresses the collusion issue but pushes the mapping burden onto the
458 service (but keeps the privacy of all information in the requestor's control).
459 The following sections describe how the WS-* specifications are used and extended to create a
460 federation framework to support these concepts.

461 2.2 Metadata Model

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

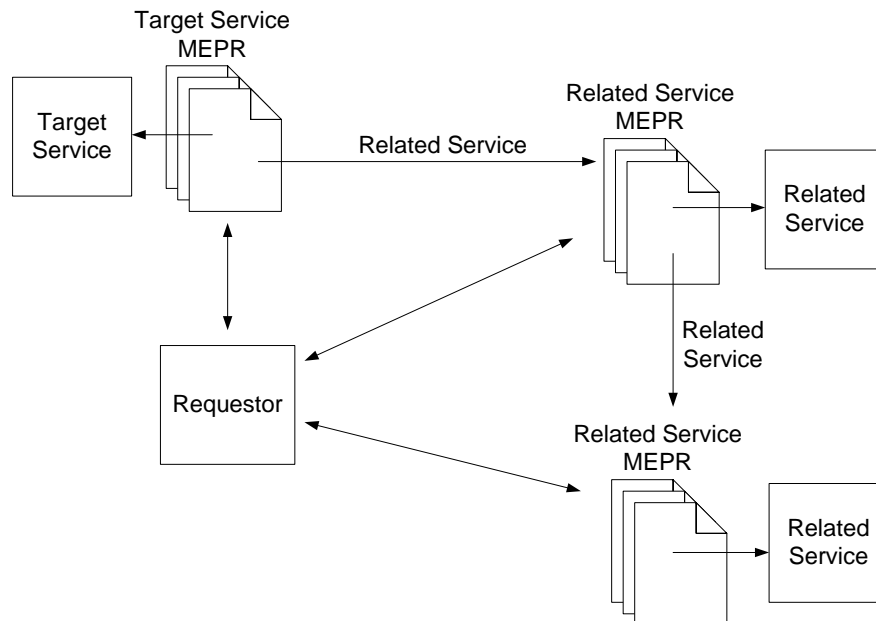
465 This discovery process begins with the target service, that is, the service to which the requester wishes to
466 ultimately communicate. Given the metadata endpoint reference (MEPR) for the target service allows the
467 requestor to obtain all requirement metadata about the service (e.g. federation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

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

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

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

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



485

486

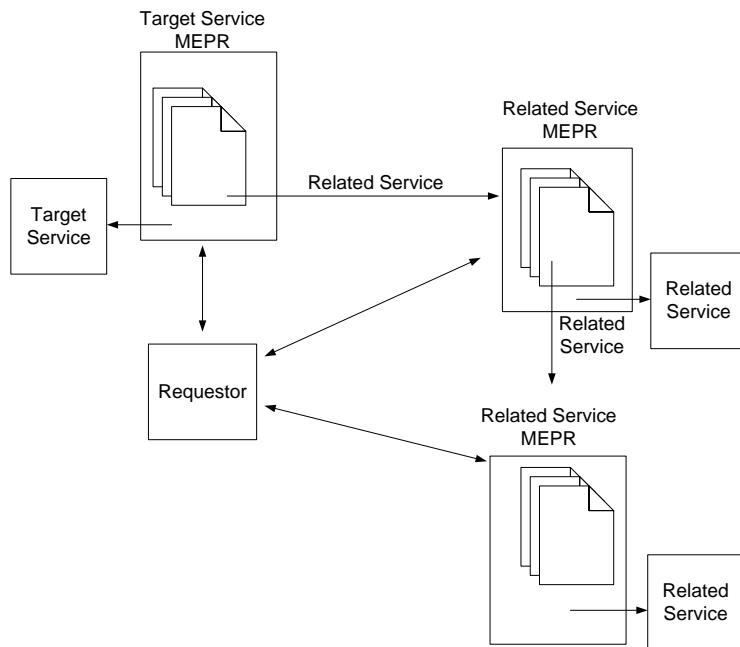
Figure 5a: Obtaining Federation Metadata (not embedded in EPR)

487
488

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

489
490
491
492

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



493

494

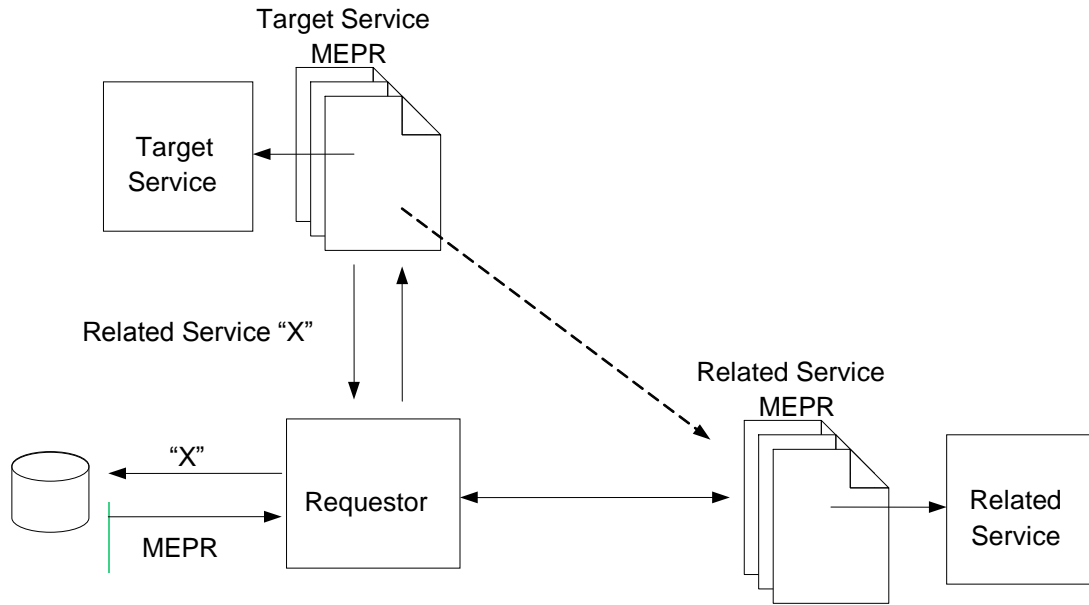
Figure 5b: Obtaining Federation Metadata (embedded)

495
496
497
498

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

499
500
501
502
503

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



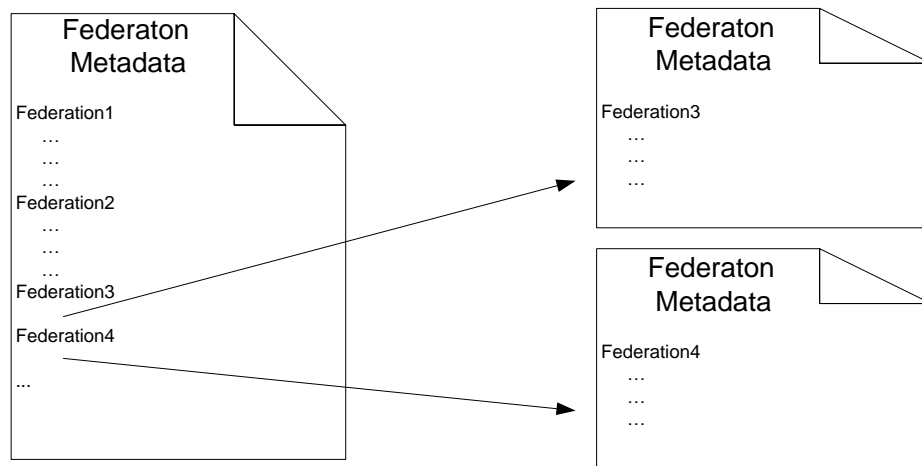
504

505

Figure 6: Example of Logical Service Names

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

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



516

517

Figure 7: Federation Metadata Document

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

521 **2.3 Security Model**

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

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

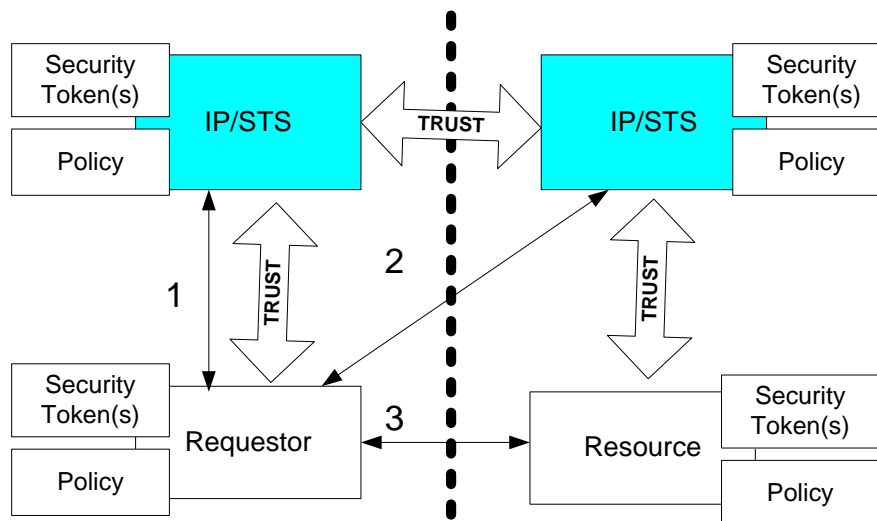
533 These initial security tokens MAY be accepted by various Web services or exchanged at Security Token
534 Services (STS) / Identity Providers (IP) for additional security tokens subject to established trust
535 relationships and trust policies as described in WS-Trust. This exchange can be used to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

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

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

543 **2.4 Trust Topologies and Security Token Issuance**

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



550

551

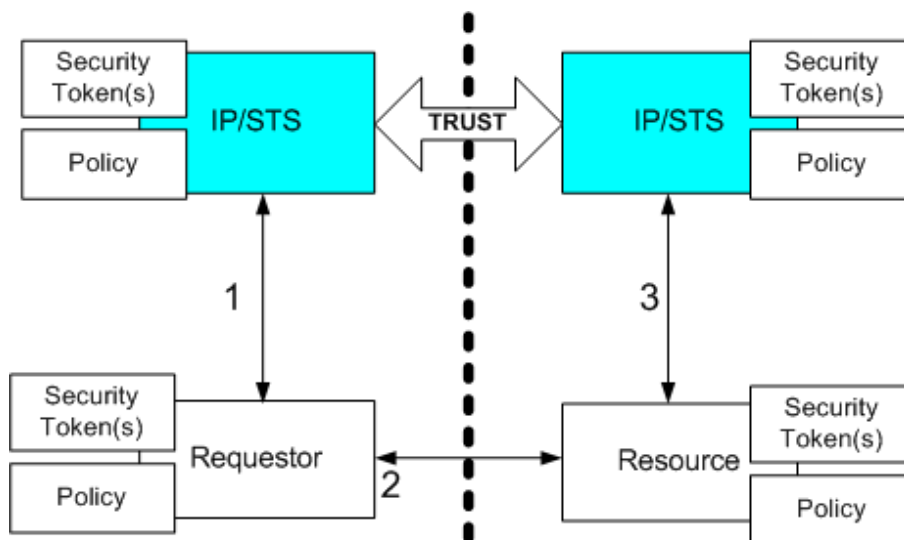
Figure 8: Federation and Trust Model

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

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

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

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



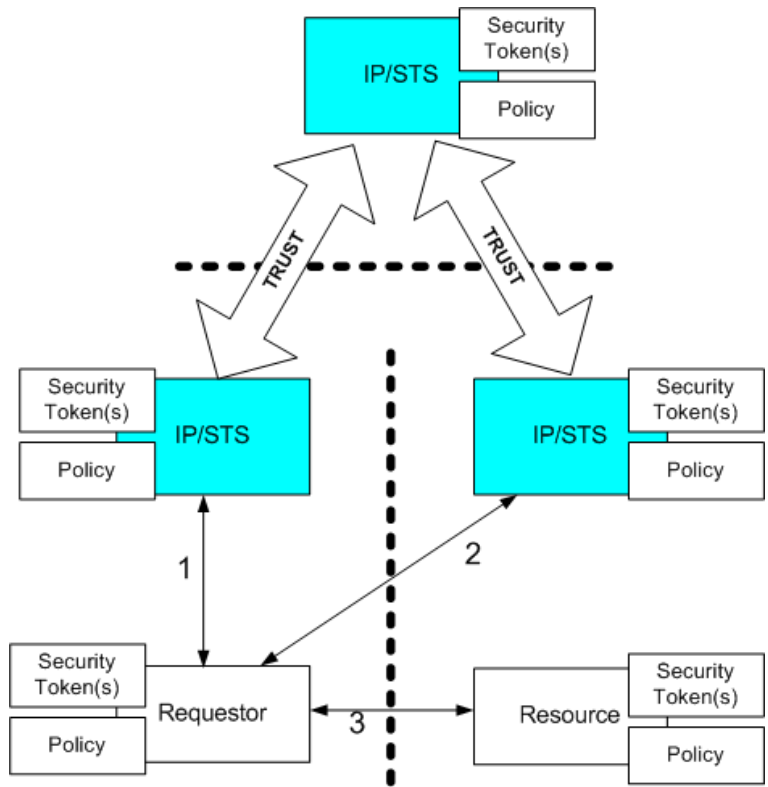
565

566

Figure 9: Alternate Federation and Trust Model

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

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

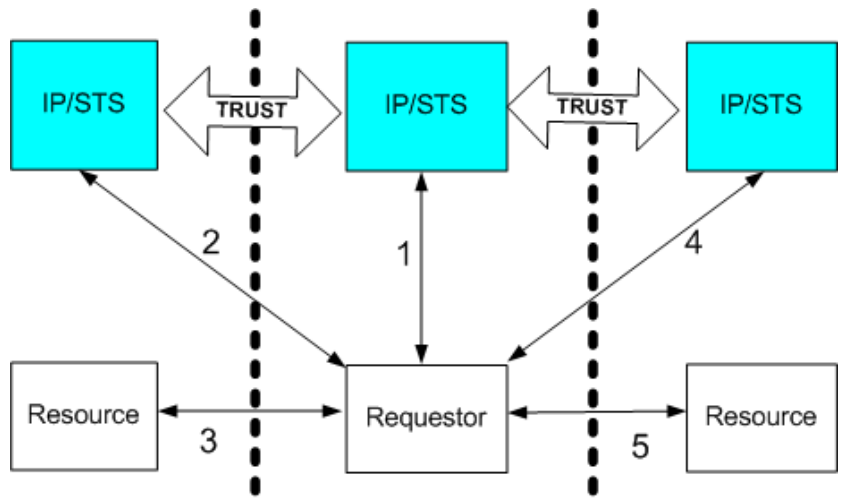


573

574

Figure 10: Indirect Trust

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

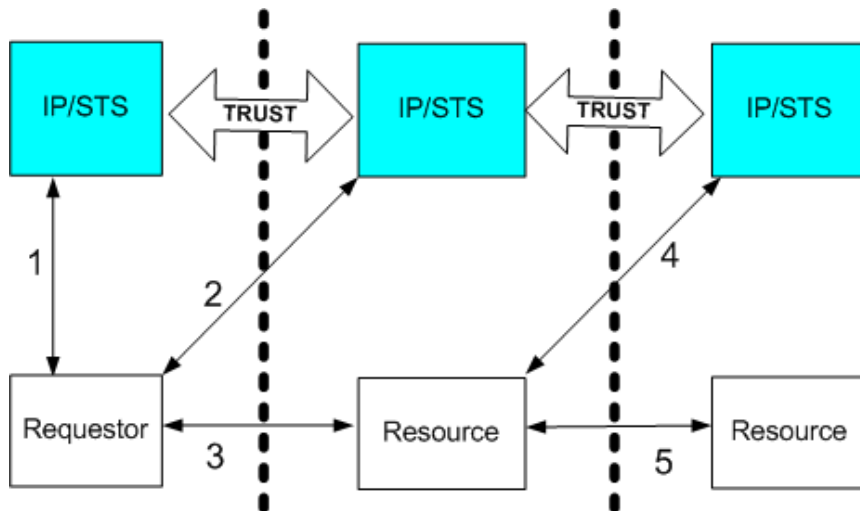


577

578

Figure 11: Multiple Trust Domains

579 Similarly, in response to a request a resource/service may need to access other resources/service on
 580 behalf of the requestor as illustrated in figure 12:

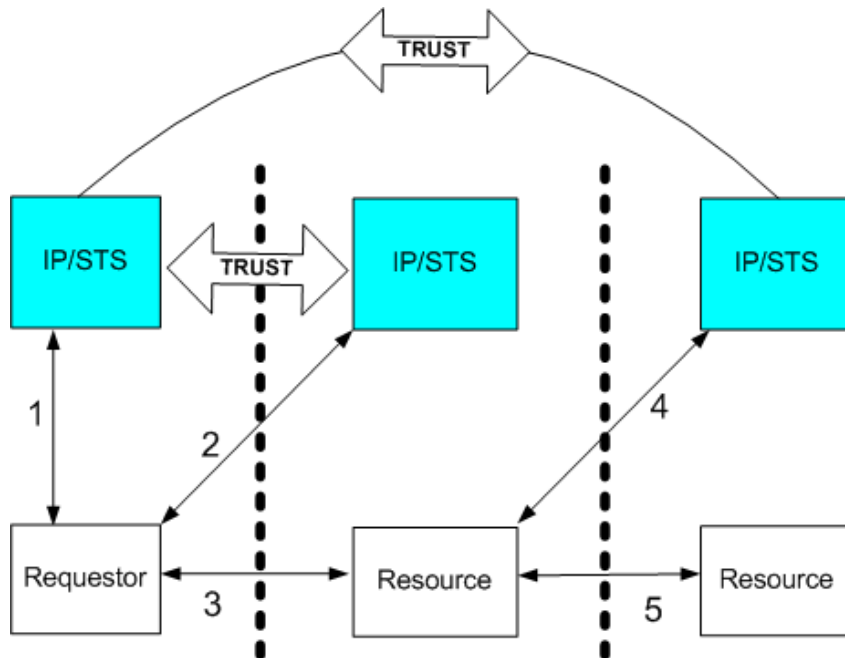


581

582

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

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



589

590

Figure 13: No Trust Relationship between Resource Providers

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

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

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

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

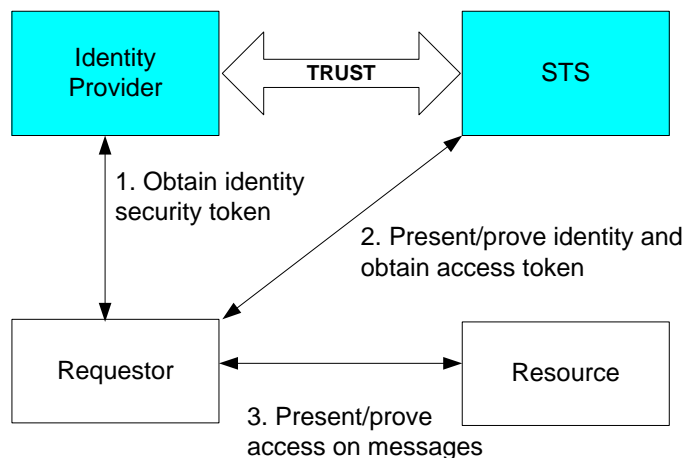
602 2.5 Identity Providers

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

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

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

613 The following example illustrates a possible combination of an Identity Provider (IP) and STS. In Figure
614 14, a requestor obtains an identity security token from its Identity Provider (1) and then presents/proves
615 this to the STS for the desired resource. If successful (2), and if trust exists and authorization is
616 approved, the STS returns an access token to the requestor. The requestor then uses the access token
617 on requests to the resource or Web service (3). Note that it is assumed that there is a trust relationship
618 between the STS and the identity provider.



619

620

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

621 2.6 Attributes and Pseudonyms

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

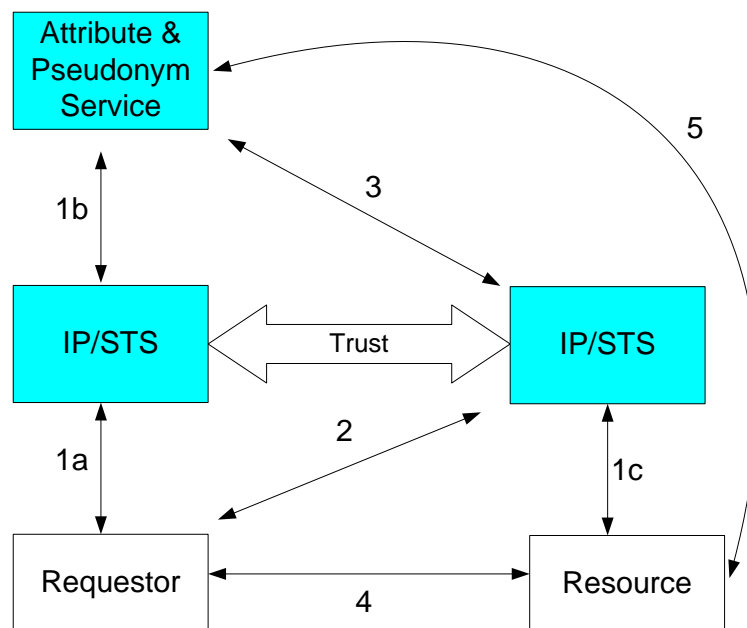
626 Protecting privacy in a federated environment often requires additional controls and mechanisms. One
627 such example is detailed access control for any information that may be considered personal or subject to
628 privacy governances. Another example is obfuscation of identity information from identity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

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

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

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

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



651

652

Figure 15: Attributes & Pseudonyms

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

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

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

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

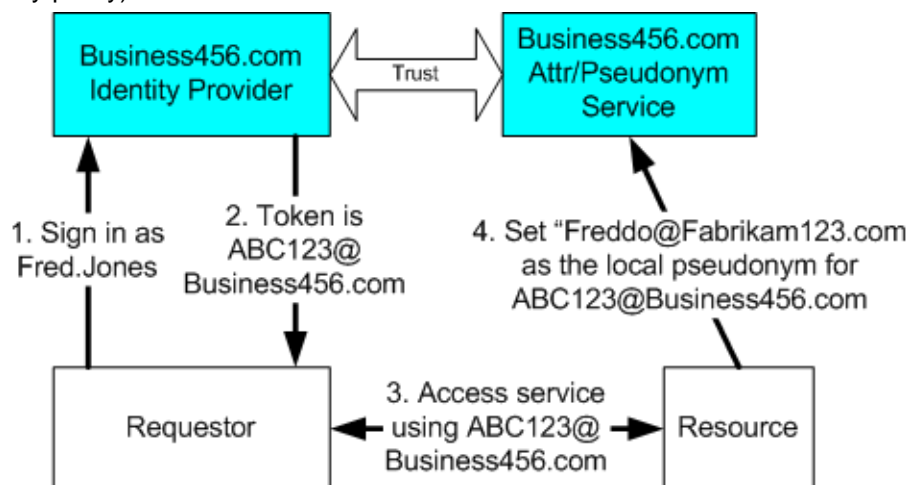
674 While pseudonyms are helpful for principals who want to keep from having their activities tracked
675 between the various sites they visit, they may add a level of complexity as the principal must typically
676 manage the authorization and privacy of each pseudonym. For principals who find this difficult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

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

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

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

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



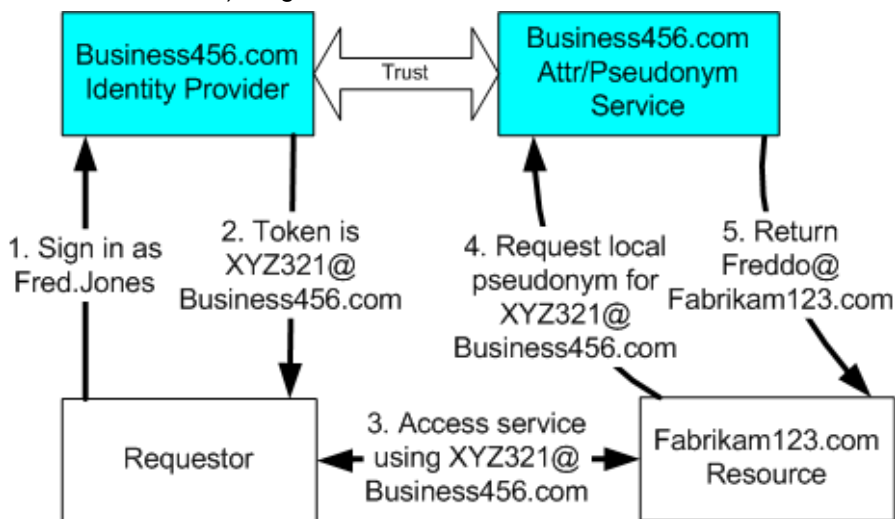
699

700

Figure 16: Pseudonym

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

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

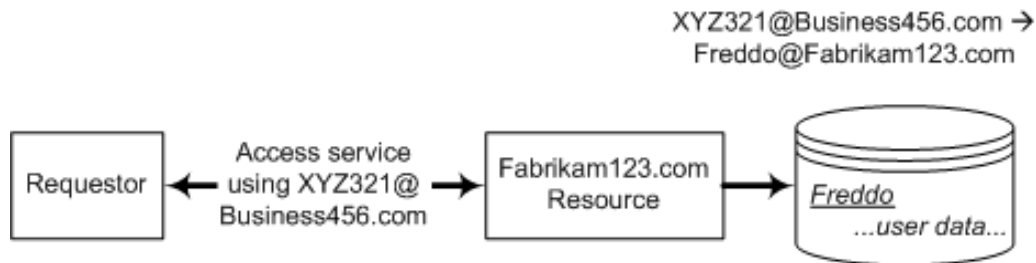


715

716

Figure 17: Pseudonym - local id

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

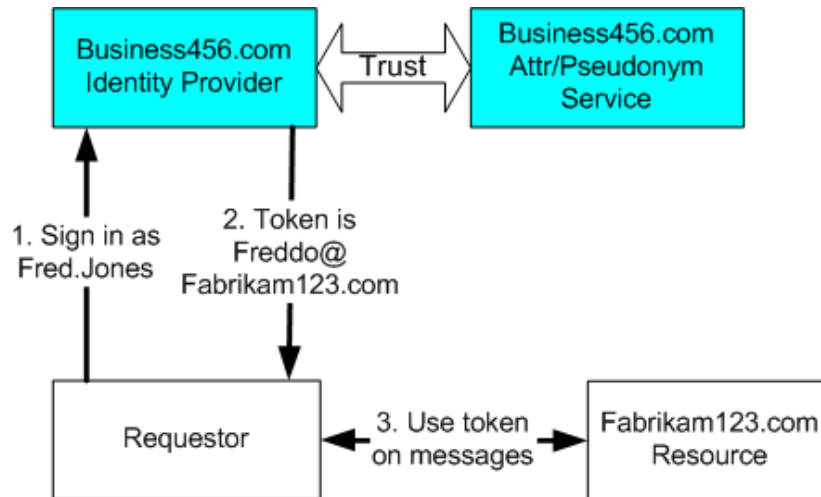


719

720

Figure 18: Pseudonym - local realm

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



726

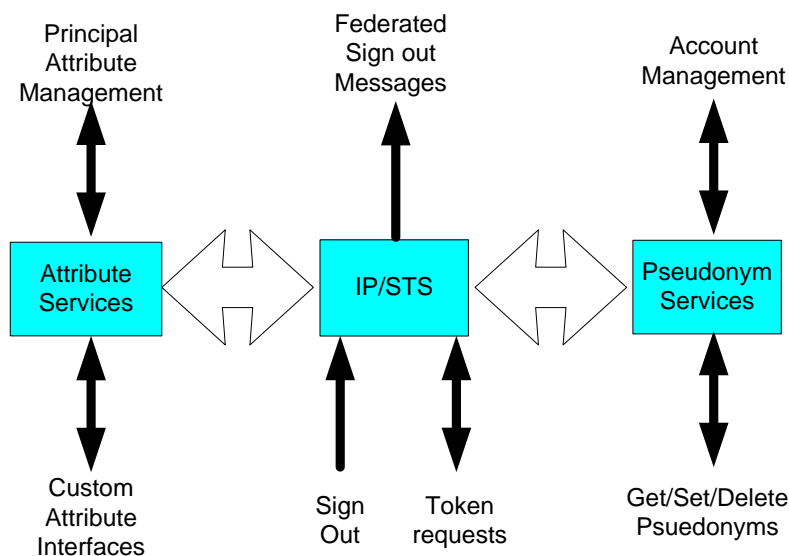
727

Figure 19: Pseudonym – token acceptance

728 The second approach is to create static identities for each service (or a group of services). That is,
 729 principle A at service X is given the digital identity 12, principle A at service Y is given the digital identity
 730 75, principle B at service X is given the digital identity 46, and so on. Operationally this approach is much
 731 like the last variation from the first approach. That is, the requestor must map its identity to an identity for
 732 the service (or service group) via a token request from its IP/STS (or using the pseudonym service
 733 directly). Consequently requestor mapping from random identities and pair-wise mapping are functionally
 734 equivalent.

735 2.7 Attributes, Pseudonyms, and IP/STS Services

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



743

744

Figure 20: Pseudonyms, Attributes and Token Issuance

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

753 3 Federation Metadata

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

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

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

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

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

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

780 3.1 Federation Metadata Document

781 The federation metadata document is an XML document containing a set of one or more OPTIONAL XML
782 elements that organizations can fill to proffer information that may be useful to partners for establishing a
783 federation. This section defines the overall document format and several OPTIONAL elements that MAY
784 be included in the federation metadata document.

785 There are two formats for the federation metadata document. The distinction between the two forms can
786 be made based on the namespace of the root element of the metadata document.

787 The federation metadata document SHOULD be of the following form:

```
788 <?xml version="1.0" encoding="..." ?>  
789 <md:EntitiesDescriptor xmlns:md="..." .../> |  
790 <md:EntityDescriptor [fed:FederationID="..."] xmlns:md="..." .../>
```

791 This form of the federation metadata document extends the core concept of the SAML metadata
792 document [Samlv2Meta] by removing the restriction that it only describes SAML entities.

793 /md:EntitiesDescriptor

794 This element is used to express authoritative information about all participants in a federation.

795 /md:EntityDescriptor

796 This element is used to express all of the metadata which a service provider chooses to publish
797 about its participation in a specific federation.

798 /md:EntityDescriptor/@fed:FederationID

799 This OPTIONAL string attribute provides an identifier for the federation to which the federation
800 metadata applies. When the metadata for a service provider is published as an
801 <md:EntityDescriptor> element of a Named <md:EntitiesDescriptor> grouping, the value of the
802 fed:FederationID attribute MUST be the same as the value of the md:Name attribute of the
803 <md:EntitiesDescriptor> element.

804

805 The federation metadata document MAY be of the following form:

```
806 <?xml version="1.0" encoding="..." ?>  
807 <fed:FederationMetadata xmlns:fed="..." ...>  
808   <fed:Federation [FederationID="..."] ...> +  
809     [Federation Metadata]  
810   </fed:Federation>  
811   [Signature]  
812 </fed:FederationMetadata>
```

813 Note that this form is provided for existing implementations and is discouraged for use in new
814 implementations. Each fed:Federation federation section in this format is functionally equivalent to the
815 RECOMMENDED md:EntityDescriptor format described above.

816 The document consists of one or more *federation* sections which describe the metadata for the endpoint
817 within a federation. The federation section MAY specify an URI indicating an identifier for the federation
818 using the `FederationID` attribute, or it MAY omit this identifier indicating the “default federation”. A
819 federation metadata document MUST NOT contain more than one default federation, that is, , only one
820 section may omit the `FederationID` attribute if multiple sections are provided.

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

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

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

Role	Applicable Metadata Statements
Any participant	mex:MetadataReference, fed:AttributeServiceEndpoints

<i>Role</i>	<i>Applicable Metadata Statements</i>
Security Token Service	md:KeyDescriptor, fed:PseudonymServiceEndpoints, fed:SingleSignOutSubscriptionEndpoints, fed:TokenTypesOffered, fed:ClaimTypesOffered, fed:AutomaticPseudonyms fed:LogicalServiceNamesOffered
Service provider / Relying Party (includes Security Token Service)	fed:TokenIssuerName, md:KeyDescriptor, fed:SingleSignoutNotificationEndpoints

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

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

836 3.1.1 Referencing Other Metadata Documents

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

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

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

```

854 <?xml version="1.0" encoding="utf-8" ?>
855 <fed:FederationMetadata xmlns:fed="..."
856     xmlns:mex="..."
857     xmlns:wsa="..."
858     xmlns:wsse="..."
859     xmlns:ds="...">
860   <fed:Federation>
861     <fed:TokenSigningKeyInfo>
862       <wsse:SecurityTokenReference>
863         <ds:X509Data>
864           <ds:X509Certificate>
865             ...
866           </ds:X509Certificate>
867         </ds:X509Data>

```

```

868     </wsse:SecurityTokenReference>
869 </fed:TokenSigningKeyInfo>
870     ...
871 </fed:Federation>
872 <fed:Federation FederationID="http://example.com/federation35532">
873   <mex:MetadataReference>
874     <wsa:Address>http://example.com/federation35532/FedMD
875     </wsa:Address>
876   </mex:MetadataReference>
877 </fed:Federation>
878 <fed:Federation FederationID="http://example.com/federation54478">
879   <mex:MetadataReference>
880     <wsa:Address>http://example.com/federation54478/FedMD
881     </wsa:Address>
882   </mex:MetadataReference>
883 </fed:Federation>
884 </fed:FederationMetadata>

```

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

891 For example, the following examples are functionally equivalent:

```

892 <?xml version="1.0" encoding="utf-8" ?>
893 <fed:FederationMetadata xmlns:fed="..." xmlns:wsse="..." xmlns:ds="...">
894   <fed:Federation FederationID="http://example.com/f1">
895     <fed:TokenSigningKeyInfo>
896       <wsse:SecurityTokenReference>
897         <ds:X509Data>
898           <ds:X509Certificate>
899             ...
900           </ds:X509Certificate>
901         </ds:X509Data>
902       </wsse:SecurityTokenReference>
903     </fed:TokenSigningKeyInfo>
904   </fed:Federation>
905   <fed:Federation FederationID="http://example.com/federation35532">
906     <fed:TokenSigningKeyInfo>
907       <wsse:SecurityTokenReference>
908         <ds:X509Data>
909           <ds:X509Certificate>
910             ...
911           </ds:X509Certificate>
912         </ds:X509Data>
913       </wsse:SecurityTokenReference>
914     </fed:TokenSigningKeyInfo>
915   </fed:Federation>
916 </fed:FederationMetadata>

```

917 and

```

918 <?xml version="1.0" encoding="utf-8" ?>
919 <fed:FederationMetadata xmlns:fed="..." xmlns:wsse="..." xmlns:ds="...">
920   <fed:Federation FederationID="http://example.com/f1">
921     <fed:TokenSigningKeyInfo>
922       <wsse:SecurityTokenReference>
923         <ds:X509Data>
924           <ds:X509Certificate>
925             ...

```

```

926         </ds:X509Certificate>
927     </ds:X509Data>
928     </wsse:SecurityTokenReference>
929 </fed:TokenSigningKeyInfo>
930 </fed:Federation>
931 <fed:Federation FederationID="http://example.com/federation35532">
932     <fed:FederationInclude>http://example.com/fl</fed:FederationInclude>
933 </fed:Federation>
934 </fed:FederationMetadata>

```

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

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

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

946 3.1.2 Role Descriptor Types

947 There are concrete service roles defined for `<md:EntityDescriptor>` which are similar to roles performed
948 by some of the WS-Federation *service instances*. The SAML `<md:IDPSSODescriptor>` element defines a
949 role similar to that of the WS-Federation `<fed:TokenIssuerEndpoints>` element and the
950 `<md:AttributeAuthorityDescriptor>` element corresponds to the `<fed:AttributeServiceEndpoints>` element.
951 There is no direct [Samlv2Meta] corollary for the WS-Federation `<fed:PseudonymServiceEndpoints>`
952 element.

953

954 The service roles for these three WS-Federation Identity Provider services, and for a generic Relying
955 Party application service, are derived from `<md:RoleDescriptor>` using the `xsi:type` extensibility
956 mechanism. For clarity schema is used in defining the following types rather than the exemplar used
957 throughout the rest of the specification.

958 3.1.2.1 WebServiceDescriptorType

959 All of the concrete role definitions of `md:EntityDescriptor` are expressed in terms of SAML profiles and
960 protocols. The `fed:WebServiceDescriptorType` is defined here as an extension of `md:RoleDescriptor` for
961 use in `md:EntityDescriptor` for the expression of WS-Federation service instances.

```

962 <complexType name="WebServiceDescriptorType" abstract="true">
963     <complexContent>
964         <extension base="md:RoleDescriptorType">
965             <sequence>
966                 <element ref="fed:LogicalServiceNamesOffered"
967                     minOccurs="0" maxOccurs="1" />
968                 <element ref="fed:TokenTypesOffered"
969                     minOccurs="0" maxOccurs="1" />
970                 <element ref="fed:ClaimDialectsOffered"
971                     minOccurs="0" maxOccurs="1" />
972                 <element ref="fed:ClaimTypesOffered"
973                     minOccurs="0" maxOccurs="1" />
974                 <element ref="fed:ClaimTypesRequested"
975                     minOccurs="0" maxOccurs="1" />

```

```

976     <element ref="fed:AutomaticPseudonyms"
977         minOccurs="0" maxOccurs="1"/>
978     <element ref="fed:TargetScopes"
979         minOccurs="0" maxOccurs="1"/>
980 </sequence>
981 <attribute name="ServiceDisplayName" type="xs:String" use="optional"/>
982 <attribute name="ServiceDescription" type="xs:String" use="optional"/>
983 </extension>
984 </complexContent>
985 </complexType>
986
987 <element name='LogicalServiceNamesOffered'
988     type=fed:LogicalServiceNamesOfferedType' />
989 <element name="fed:TokenTypeOffered" type="fed:TokenType"/>
990 <element name="fed:ClaimDialectsOffered" type="fed:ClaimDialectsOfferedType"/>
991 <element name="fed:ClaimTypesOffered" type="fed:ClaimTypesOfferedType"/>
992 <element name="ClaimTypesRequested" type="tns:ClaimTypesRequestedType"/>
993 <element name="fed:AutomaticPseudonyms" type="xs:boolean"/>
994 <element name="fed:TargetScope" type="tns:EndpointType"/>

```

995

996 /fed:WebServiceDescriptor/@ServiceDisplayName

997 This OPTIONAL string attribute provides a friendly name for this service instance that can be
998 shown in user interfaces. It is a human readable label that can be used to index metadata
999 provided for different service instances.

1000 /fed:WebServiceDescriptor/@ServiceDescription

1001 This OPTIONAL string attribute provides a description for this service instance that can be shown
1002 in user interfaces. It is a human readable description that can be used to understand the type of
1003 service to which the metadata applies.

1004 /fed:WebServiceDescriptor/fed:LogicalServiceNamesOffered

1005 This OPTIONAL element allows a federation metadata provider to specify to specify a “logical
1006 name” that is associated with the service. See section 3.1.3 details.

1007 /fed:WebServiceDescriptor/fed:TokenTypeOffered

1008 This OPTIONAL element allows a federation metadata provider to specify token types that can be
1009 issued by the service. See section 3.1.8 for details.

1010 /fed:WebServiceDescriptor/fed:ClaimTypesOffered

1011 This OPTIONAL element allows a federation metadata provider to specify offered claim types,
1012 using the schema provided by the common claim dialect defined in this specification that can be
1013 asserted in security tokens issued by the service. See section 3.1.9 for details.

1014 /fed:WebServiceDescriptorType/fed:ClaimTypeRequested

1015 This OPTIONAL element allows a federation metadata provider to specify claim types, using the
1016 schema provided by the common claim dialect defined in this specification, that MAY or MUST be
1017 present in security tokens requested by the service. See section 3.1.10 for additional details.

1018 /fed:WebServiceDescriptor/fed:ClaimDialectsOffered

1019 This OPTIONAL element allows a federation metadata provider to specify dialects, via URI(s),
1020 that are accepted in token requests to express the syntax for requested claims. See section
1021 3.1.11 for details.

1022 /fed:WebServiceDescriptor/fed:AutomaticPseudonyms

1023 This OPTIONAL element allows a federation metadata provider to indicate if it automatically
1024 maps pseudonyms or applies some form of identity mapping. See section 3.1.12 for details.

1025 /fed:WebServiceDescriptor/fed:TargetScope

1026 This OPTIONAL element allows a federation metadata provider to indicate the EPRs that are
1027 associated with token scopes of the relying party or STS. See section 3.1.14 for details.

1028

1029 New complex service types for Security Token, Attribute and Pseudonym services are derived from
1030 fed:WebServiceDescriptorType as described in the following sections. These types will be used to
1031 extend <md:RoleDescriptor> to create service roles which are similar to <md:IDPSSODescriptor>. A
1032 new complex generic application service type is also derived from fed:WebServiceDescriptorType . This
1033 type will be used to extend <md:RoleDescriptor> to create a service role which is similar to
1034 <md:SPSSODescriptor>.

1035 3.1.2.2 SecurityTokenServiceType

```
1036 <complexType name="SecurityTokenServiceType">  
1037   <extension base="fed:WebServiceDescriptorType">  
1038     <sequence>  
1039       <element ref="fed:SecurityTokenServiceEndpoint"  
1040         minOccurs="1" maxOccurs="unbounded"/>  
1041       <element ref="fed:SingleSignOutSubscriptionEndpoint"  
1042         minOccurs="0" maxOccurs="unbounded"/>  
1043       <element ref="fed:SingleSignOutNotificationEndpoint"  
1044         minOccurs="0" maxOccurs="unbounded"/>  
1045       <element ref="fed:PassiveRequestorEndpoint"  
1046         minOccurs="0" maxOccurs="unbounded"/>  
1047     </sequence>  
1048   </extension>  
1049 </complexType>  
1050 <element name="fed:SecurityTokenServiceEndpoint"  
1051   type="wsa:EndpointReferenceType"/>  
1052 <element name="fed:SingleSignOutSubscriptionEndpoint"  
1053   type="wsa:EndpointReferenceType"/>  
1054 <element name="fed:SingleSignOutNotificationEndpoint"  
1055   type="wsa:EndpointReferenceType"/>  
1056 <element name="fed:PassiveRequestorEndpoint"  
1057   type="wsa:EndpointReferenceType"/>
```

1058 These definitions apply to the derived type listed in the schema outlined above.

1059 fed:SecurityTokenServiceType/fed:SecurityTokenServiceEndpoint

1060 This required element specifies the endpoint address of a security token service that supports the
1061 WS-Federation and WS-Trust interfaces. Its contents MUST be an endpoint reference as defined
1062 by [WS-Addressing] that provides a transport address for the security token service. It MAY be
1063 repeated for different, but functionally equivalent, endpoints of the same logical *service instance*.

1064 fed:SecurityTokenServiceType/fed:SingleSignOutSubscriptionServiceEndpoint

1065 This optional element specifies the endpoint address of a service which can be used to subscribe
1066 to federated sign-out messages. Its contents MUST be an endpoint reference as defined by [WS-
1067 Addressing] that provides a transport address for the subscription service. It MAY be repeated
1068 for different, but functionally equivalent, endpoints of the same logical *service instance*.

1069 fed:SecurityTokenServiceType/fed:SingleSignOutNotificationServiceEndpoint

1070 This optional element specifies the endpoint address of a service to which push notifications of
1071 sign-out are to be sent. Its contents MUST be an endpoint reference as defined by [WS-
1072 Addressing] that provides a transport address for the notification service. It MAY be repeated for
1073 different, but functionally equivalent, endpoints of the same logical *service instance*.

1074 fed:SecurityTokenServiceType/fed:PassiveRequestorEndpoint

1075 This optional element specifies the endpoint address of a service that supports the WS-
1076 Federation Web (Passive) Requestor protocol. It MAY be repeated for different, but functionally
1077 equivalent, endpoints of the same logical *service instance*.

1078

1079 An <md:EntityDescriptor> that provides a WS-Federation based security token service is indicated by
1080 using the <md:RoleDescriptor> extensibility point as follows.

1081

```
1082 <EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"  
1083   xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"  
1084   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
1085   entityID="...">  
1086   <ds:Signature>...</ds:Signature>  
1087   <RoleDescriptor xsi:type="fed:SecurityTokenServiceType"  
1088     protocolSupportEnumeration="http://docs.oasis-  
1089 open.org/wsfed/federation/200706"  
1090     "http://docs.oasis-open.org/ws-sx/ws-trust/200512">  
1091     ...  
1092   </RoleDescriptor>  
1093   ...  
1094 </EntityDescriptor>
```

1095

1096 3.1.2.3 PseudonymServiceType

```
1097 <complexType name="PseudonymServiceType">  
1098   <extension base="fed:WebServiceDescriptorType">  
1099     <sequence>  
1100       <element ref="fed:PseudonymServiceEndpoint"  
1101         minOccurs="1" maxOccurs="unbounded"/>  
1102       <element ref="fed:SingleSignOutNotificationEndpoint"  
1103         minOccurs="0" maxOccurs="unbounded"/>  
1104     </sequence>  
1105   </extension>  
1106 </complexType>  
1107 <element name="fed:PseudonymServiceEndpoint"  
1108   type="tns:EndpointType"/>  
1109 <element name="fed:SingleSignOutNotificationEndpoint"  
1110   type="tns:EndpointType"/>
```

1111 These definitions apply to the derived type listed in the schema outlined above.

1112 fed:PseudonymServiceType/fed:PseudonymServiceEndpoint

1113 This required element specifies the endpoint address of a pseudonym service that supports the
1114 WS-Federation and WS-Trust interfaces. Its contents MUST be an endpoint reference as defined
1115 by [WS-Addressing] that provides a transport address for the pseudonym service. It MAY be
1116 repeated for different, but functionally equivalent, endpoints of the same logical *service instance*.

1117 fed:PseudonymServiceType/fed:SingleSignOutNotificationServiceEndpoint

1118 This optional element specifies the endpoint address of a service to which push notifications of
1119 sign-out are to be sent. Its contents MUST be an endpoint reference as defined by [WS-
1120 Addressing] that provides a transport address for the notification service. It MAY be repeated for
1121 different, but functionally equivalent, endpoints of the same logical *service instance*.

1122

1123 An <md:EntityDescriptor> that provides a WS-Federation based pseudonym service is indicated by using
1124 the <md:RoleDescriptor> extensibility point as follows.

1125

```

1126 <EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
1127   xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
1128   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1129   entityID="...">
1130   <ds:Signature>...</ds:Signature>
1131   <RoleDescriptor xsi:type="fed:PseudonymServiceType"
1132     protocolSupportEnumeration="http://docs.oasis-
1133 open.org/wsfed/federation/200706"
1134     "http://docs.oasis-open.org/ws-sx/ws-trust/200512">
1135     ...
1136   </RoleDescriptor>
1137   ...
1138 </EntityDescriptor>

```

1139 3.1.2.4 AttributeServiceType

```

1140 <complexType name="AttributeServiceType">
1141   <extension base="fed:WebServiceDescriptorType">
1142     <sequence>
1143       <element ref="fed:AttributeServiceEndpoint"
1144         minOccurs="1" maxOccurs="unbounded"/>
1145       <element ref="fed:SingleSignOutNotificationEndpoint"
1146         minOccurs="0" maxOccurs="unbounded"/>
1147     </sequence>
1148   </extension>
1149 </complexType>
1150 <element name="fed:AttributeServiceEndpoint"
1151   type="tns:EndpointType"/>
1152 <element name="fed:SingleSignOutNotificationEndpoint"
1153   type="tns:EndpointType"/>

```

1154 These definitions apply to the derived type listed in the schema outlined above.

1155 fed:AttributeServiceType/fed:AttributeServiceEndpoint

1156 This required element specifies the endpoint address of an attribute service that supports the
1157 WS-Federation and WS-Trust interfaces. Its contents MUST be an endpoint reference as defined
1158 by [WS-Addressing] that provides a transport address for the attribute service. It MAY be
1159 repeated for different, but functionally equivalent, endpoints of the same logical *service instance*.

1160 fed:AttributeServiceType/fed:SingleSignOutNotificationServiceEndpoint

1161 This optional element specifies the endpoint address of a service to which push notifications of
1162 sign-out are to be sent. Its contents MUST be an endpoint reference as defined by [WS-
1163 Addressing] that provides a transport address for the notification service. It MAY be repeated for
1164 different, but functionally equivalent, endpoints of the same logical *service instance*.

1165

1166 An <md:EntityDescriptor> that provides a WS-Federation based attribute service is indicated by using the
1167 <md:RoleDescriptor> extensibility point as follows.

1168

```

1169 <EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
1170   xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
1171   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1172   entityID="...">
1173   <ds:Signature>...</ds:Signature>
1174   <RoleDescriptor xsi:type="fed:AttributeServiceType"
1175     protocolSupportEnumeration="http://docs.oasis-
1176 open.org/wsfed/federation/200706"
1177     "http://docs.oasis-open.org/ws-sx/ws-trust/200512">
1178     ...

```

1179
1180
1181

```
</RoleDescriptor>  
...  
</EntityDescriptor>
```

1182

1183 3.1.2.5 ApplicationServiceType

1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201

```
<complexType name="ApplicationServiceType" <extension  
base="fed:WebServiceDescriptorType">  
  <sequence>  
    <element ref="fed:ApplicationServiceEndpoint"  
      minOccurs="1" maxOccurs="unbounded"/>  
    <element ref="fed:SingleSignOutNotificationEndpoint"  
      minOccurs="0" maxOccurs="unbounded"/>  
    <element ref="fed:PassiveRequestorEndpoint"  
      minOccurs="0" maxOccurs="unbounded"/>  
  </sequence>  
</extension>  
</complexType>  
<element name="fed:ApplicationServiceEndpoint"  
  type="tns:EndpointType"/>  
<element name="fed:SingleSignOutNotificationEndpoint"  
  type="tns:EndpointType"/>  
<element name="fed:PassiveRequestorEndpoint"  
  type="tns:EndpointType"/>
```

1202 These definitions apply to the derived type listed in the schema outlined above.

1203 fed:ApplicationServiceType/fed:ApplicationServiceEndpoint

1204 This required element specifies the endpoint address of a Relying Party application service that
1205 supports the WS-Federation and WS-Trust interfaces. Its contents MUST be an endpoint reference
1206 as defined by [WS-Addressing] that provides a transport address for the application service. It
1207 MAY be repeated for different, but functionally equivalent, endpoints of the same logical *service*
1208 *instance*.

1209 fed:ApplicationServiceType/fed:SingleSignOutNotificationServiceEndpoint

1210 This optional element specifies the endpoint address of a service to which push notifications of
1211 sign-out are to be sent. Its contents MUST be an endpoint reference as defined by [WS-
1212 Addressing] that provides a transport address for the notification service. It MAY be repeated for
1213 different, but functionally equivalent, endpoints of the same logical *service instance*.

1214 fed:ApplicationServiceType/fed:PassiveRequestorEndpoint

1215 This optional element specifies the endpoint address of a service that supports the WS-
1216 Federation Web (Passive) Requestor protocol. It MAY be repeated for different, but functionally
1217 equivalent, endpoints of the same logical *service instance*.

1218

1219 An <md:EntityDescriptor> that provides a WS-Federation based application service is indicated by using
1220 the <md:RoleDescriptor> extensibility point as follows.

1221

1222
1223
1224
1225
1226
1227
1228
1229

```
<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"  
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"  
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
  entityID="...">  
  <ds:Signature>...</ds:Signature>  
  <RoleDescriptor xsi:type="fed:ApplicationServiceType"  
    protocolSupportEnumeration="http://docs.oasis-  
open.org/ws-federation/200706"
```

1230
1231
1232
1233
1234

1235

```
        "http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        ...
    </RoleDescriptor>
    ...
</EntityDescriptor>
```

1236 3.1.3 LogicalServiceNamesOffered Element

1237 In some scenarios token issuers are referred to be a logical name representing an equivalence class of
1238 issuers. For example, a Relying Party may not care what specific bank issues a token so long as the
1239 issuance is associated with a specific credit card program. To facilitate this, federated metadata provides
1240 the <sp:TokenIssuerName> element (described in [WS-SecurityPolicy]) to indicate that a Relying Party
1241 needs a token from a specific class of issuer.

1242 As stated, the OPTIONAL <fed:LogicalServiceNamesOffered> element allows a federation
1243 metadata provider, specifically a token service in this case, to specify a set of “logical names” that are
1244 associated with the provider. That is, when a Relying Party indicates a logical name for a token issuer
1245 using the <sp:TokenIssuerName> element in a token assertion the
1246 <fed:LogicalServiceNamesOffered> element this element can be used as a correlation
1247 mechanism by clients. This element populates the [Federation Metadata] property. This is typically a
1248 service-level statement but can be an endpoint-level statement.

1249 The schema for this optional element is shown below.

```
1250 <fed:LogicalServiceNamesOffered ...>
1251   <fed:IssuerName Uri="xs:anyURI" .../> +
1252 </fed:LogicalServiceNamesOffered>
```

1253 The following example illustrates using this optional element to specify a logical name of the federating
1254 organization as a token issuer.

```
1255 <fed:LogicalServiceNamesOffered>
1256   <fed:IssuerName Uri="http://fabrikam.com/federation/corporate" />
1257 </fed:LogicalServiceNamesOffered>
```

1258

1259 3.1.4 PseudonymServiceEndpoints Element

1260 The OPTIONAL <fed:PseudonymServiceEndpoints> element allows a federation metadata provider
1261 to specify the endpoint address of its pseudonym service (or addresses for functionally equivalent
1262 pseudonym services) which can be referenced by federated partners when requesting tokens from it.
1263 When present, this indicates that services SHOULD use the pseudonym service to map identities to local
1264 names as the identities MAY vary across invocations. This element populates the [Federation Metadata]
1265 property. This is typically specified by token issuers and security token services. This is typically a
1266 service-level statement but can be an endpoint-level statement.

1267 The schema for this optional element is shown below.

```
1268 <fed:PseudonymServiceEndpoints>
1269   wsa:EndpointReferenceType +
1270 </fed:PseudonymServiceEndpoints>
```

1271 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1272 a transport address for an STS interface to the pseudonym service (or functionally equivalent pseudonym
1273 service endpoints). Each endpoint reference MAY (and SHOULD if there is no expectation that the policy
1274 is known *a priori*) include metadata for the STS endpoint or a reference to an endpoint from where such

1275 metadata can be retrieved by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for
1276 additional details).

1277 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1278 semantics defined in this specification.

1279 It should be noted that this element MAY occur multiple times indicating distinct services with different
1280 capabilities. Service providers MUST include equivalent endpoints – different endpoint references for a
1281 single service, or for a set of logically equivalent services – in a single
1282 <fed:PseudonymServiceEndpoints> element.

1283 The following example illustrates using this optional element to specify an endpoint address for the
1284 pseudonym service of the federating organization.

```
1285 <fed:PseudonymServiceEndpoints>  
1286   <wsa:Address> http://fabrkam.com/federation/Pseudo </wsa:Address>  
1287 </fed:PseudonymServiceEndpoints>
```

1288 **3.1.5 AttributeServiceEndpoints Element**

1289 The OPTIONAL <fed:AttributeServiceEndpoints> element allows a federation metadata
1290 provider to specify the endpoint address of its attribute service (or addresses for functionally equivalent
1291 attribute services) which can be referenced by federated partners when requesting tokens from it. This
1292 element populates the [Federation Metadata] property. This is typically specified by requestors and is a
1293 service-level statement.

1294 The schema for this optional element is shown below.

```
1295 <fed:AttributeServiceEndpoints>  
1296   wsa:EndpointReferenceType +  
1297 </fed:AttributeServiceEndpoints>
```

1298 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1299 a transport address for an STS interface to the service (or functionally equivalent attribute service
1300 endpoints). Each endpoint reference MAY (and SHOULD if there is no expectation that the policy is
1301 known *a priori*) include metadata for the STS endpoint or a reference to an endpoint from where such
1302 metadata can be retrieved by a token requestor (see [WS-Addressing] and [WS-MetadataExchange] for
1303 additional details).

1304 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1305 semantics defined in this specification.

1306 It should be noted that this element MAY occur multiple times indicating distinct services with different
1307 capabilities. Service providers MUST include equivalent endpoints – different endpoint references for a
1308 single service, or for a set of logically equivalent services – in a single <fed:AttributeServiceEndpoints>
1309 element.

1310 The following example illustrates using this optional element to specify an endpoint address for the
1311 attribute service of the federating organization.

```
1312 <fed:AttributeServiceEndpoints>  
1313   <wsa:Address> http://fabrkam.com/federation/Attr </wsa:Address>  
1314 </fed:AttributeServiceEndpoints>
```

1315 **3.1.6 SingleSignOutSubscriptionEndpoints Element**

1316 The OPTIONAL <fed:SingleSignOutSubscriptionEndpoints> element allows a federation
1317 metadata provider to specify the endpoint address of its subscription service (or addresses for functionally
1318 equivalent subscription services) which can be used to subscribe to federated sign-out messages. This

1319 element populates the [Federation Metadata] property. This is typically specified by token issuers and
1320 security token services. This is typically a service-level statement but can be an endpoint-level statement.
1321 The schema for this optional element is shown below.

```
1322 <fed:SingleSignOutSubscriptionEndpoints>  
1323   wsa:EndpointReferenceType +  
1324 </fed:SingleSignOutSubscriptionEndpoints>
```

1325 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1326 a transport address for the subscription manager (or functionally equivalent subscription services).
1327 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1328 semantics defined in this specification.

1329 **3.1.7 SingleSignOutNotificationEndpoints Element**

1330 Services MAY subscribe for sign-out notifications however clients MAY also push notifications to services.
1331 The OPTIONAL <fed:SingleSignOutNotificationEndpoints> element allows a federation
1332 metadata provider to specify the endpoint address (or functionally equivalent addresses) to which push
1333 notifications of sign-out are to be sent. This element populates the [Federation Metadata] property. This
1334 is typically specified by service providers and security token services. This is typically a service-level
1335 statement but can be an endpoint-level statement.

1336 The schema for this optional element is shown below.

```
1337 <fed:SingleSignOutNotificationEndpoints>  
1338   wsa:EndpointReferenceType +  
1339 </fed:SingleSignOutNotificationEndpoints>
```

1340 The content of this element is one, or more, endpoint references as defined by [WS-Addressing] providing
1341 a transport address for the notification service (or functionally equivalent notification service endpoints) .

1342 This element allows attributes to be added. Use of this extensibility point MUST NOT alter the
1343 semantics defined in this specification.

1344 **3.1.8 TokenTypesOffered Element**

1345 The OPTIONAL <fed:TokenTypesOffered> element allows a federation metadata provider to specify
1346 the list of offered security token types that can be issued by its STS. A federated partner can use the
1347 offered token types to decide what token type to ask for when requesting tokens from it. This element
1348 populates the [Federation Metadata] property. This is typically specified by token issuers and security
1349 token services. This is typically a service-level statement but can be an endpoint-level statement.

1350 The schema for this optional element is shown below.

```
1351 <fed:TokenTypesOffered ...>  
1352   <fed:TokenType Uri="xs:anyURI" ...>  
1353     ...  
1354   </fed:TokenType> +  
1355   ...  
1356 </fed:TokenTypesOffered>
```

1357 The following describes the elements listed in the schema outlined above:

1358 /fed: TokenTypesOffered

1359 This element is used to express the list of token types that the federating STS is capable of
1360 issuing.

1361 /fed:TokenTypesOffered/fed:TokenType

1362 This element indicates an individual token type that the STS can issue.

- 1363 /fed:TokenTypesOffered/fed:TokenType/@Uri
 1364 This attribute provides the unique identifier (URI) of the individual token type that the STS can
 1365 issue.
- 1366 /fed:TokenTypesOffered/fed:TokenType/{any}
 1367 The semantics of any content for this element are undefined. Any extensibility or use of sub-
 1368 elements MUST NOT alter the semantics defined in this specification.
- 1369 /fed:TokenTypesOffered/fed:TokenType/@{any}
 1370 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
 1371 MUST NOT violate or alter the semantics defined in this specification.
- 1372 /fed:TokenTypesOffered/@{any}
 1373 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
 1374 MUST NOT violate or alter the semantics defined in this specification.
- 1375 /fed:TokenTypesOffered/{any}
 1376 The semantics of any content for this element are undefined. Any extensibility or use of sub-
 1377 elements MUST NOT alter the semantics defined in this specification.
- 1378 The following example illustrates using this optional element to specify that the issuing STS of the
 1379 federating organization can issue both SAML 1.1 and SAML 2.0 tokens [WSS:SAMLTokenProfile].

```
1380 <fed:TokenTypesOffered>
1381   <fed:TokenType Uri="urn:oasis:names:tc:SAML:1.1" />
1382   <fed:TokenType Uri="urn:oasis:names:tc:SAML:2.0" />
1383 </fed:TokenTypesOffered>
```

1384 3.1.9 ClaimTypesOffered Element

- 1385 The OPTIONAL `<fed:ClaimTypesOffered>` element allows a federation metadata provider such as
 1386 an IdP to specify the list of publicly offered claim types, named using the schema provided by the
 1387 common claims dialect defined in this specification, that can be asserted in security tokens issued by its
 1388 STS. It is out of scope of this specification whether or not a URI used to name a claim type resolves.
 1389 Note that issuers MAY support additional claims and that not all claims may be available for all token
 1390 types. If other means of describing/identifying claims are used in the future, then corresponding XML
 1391 elements can be introduced to publish the new claim types. A federated partner can use the offered claim
 1392 types to decide which claims to ask for when requesting tokens from it. This specification places no
 1393 requirements on the syntax used to describe the claims. This element populates the [Federation
 1394 Metadata] property. This is typically specified by token issuers and security token services. This is
 1395 typically a service-level statement but can be an endpoint-level statement.
- 1396 The schema for this optional element is shown below.

```
1397 <fed:ClaimTypesOffered ...>
1398   <auth:ClaimType ...> ... </auth:ClaimType> +
1399 </fed:ClaimTypesOffered>
```

- 1400 The following describes the elements listed in the schema outlined above:

- 1401 /fed:ClaimTypesOffered
 1402 This element is used to express the list of claim types that the STS is capable of issuing.
- 1403 /fed:ClaimTypesOffered/@{any}
 1404 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
 1405 NOT alter the semantics defined in this specification.

1406 The following example illustrates using this optional element to specify that the issuing STS of the
1407 federating organization can assert two claim types named using the common claims format.

```
1408 <fed:ClaimTypesOffered>  
1409   <auth:ClaimType Uri="http://.../claims/EmailAddr" >  
1410     <auth:DisplayName>Email Address</auth:DisplayName>  
1411   </auth:ClaimType>  
1412   <auth:ClaimType Uri="http://.../claims/IsMember" >  
1413     <auth:DisplayName>Is a Member (yes/no)</auth:DisplayName>  
1414     <auth:Description>If a person is a member of this club</auth:Description>  
1415   </auth:ClaimType>  
1416 </fed:ClaimTypesOffered>
```

1417 3.1.10 ClaimTypesRequested Element

1418 The OPTIONAL `<fed:ClaimTypesRequested>` element allows a federation metadata provider such
1419 as an RP to specify the list of publicly requested claim types, named using the schema provided by the
1420 common claims dialect defined in this specification, that are necessary to be asserted in security tokens
1421 used to access its services. It is out of scope of this specification whether or not a URI used to name a
1422 claim type resolves. Note that federation metadata provider MAY support additional claims and that not all
1423 claims may be available for all token types. If other means of describing/identifying claims are used in the
1424 future, then corresponding XML elements can be introduced to request the new claim types. A federated
1425 partner can use the requested claim types to decide which claims to ask for when requesting tokens for
1426 the federation metadata provider. This specification places no requirements on the syntax used to
1427 describe the claims. This element populates the [Federation Metadata] property. This is typically
1428 specified by token issuers and security token services. This is typically a service-level statement but can
1429 be an endpoint-level statement.

1430 The schema for this optional element is shown below.

```
1431 <fed:ClaimTypesRequested ...>  
1432   <auth:ClaimType ...> ... </auth:ClaimType> +  
1433 </fed:ClaimTypesRequested>
```

1434 The following describes the elements listed in the schema outlined above:

1435 `/fed:ClaimTypesRequested`

1436 This element is used to express the list of claim types that MAY or MUST be present in security
1437 tokens submitted to the service.

1438 `/fed:ClaimTypesOffered/@{any}`

1439 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
1440 NOT alter the semantics defined in this specification.

1441 The following example illustrates using this optional element to specify that the federation metadata
1442 provider requests two claim types, named using the common claims format.

```
1443 <fed:ClaimTypesRequested>  
1444   <auth:ClaimType Uri="http://.../claims/EmailAddr" >  
1445     <auth:DisplayName>Email Address</auth:DisplayName>  
1446   </auth:ClaimType>  
1447   <auth:ClaimType Uri="http://.../claims/IsMember" >  
1448     <auth:DisplayName>Is a Member (yes/no)</auth:DisplayName>  
1449     <auth:Description>If a person is a member of this club</auth:Description>  
1450   </auth:ClaimType>  
1451 </fed:ClaimTypesRequested>
```

1452

1453 3.1.11 ClaimDialectsOffered Element

1454 The OPTIONAL `fed:ClaimDialectsOffered` element allows a federation metadata provider to specify the
1455 list of dialects, named using URIs, that are accepted by its STS in token requests to express the claims
1456 requirement. A federated partner can use is list to decide which dialect to use to express its desired
1457 claims when requesting tokens from it. This specification defines one standard claims dialect in the
1458 subsequent section 9.3, but other claim dialects MAY be defined elsewhere for use in other scenarios.
1459 This element populates the [Federation Metadata] property. This is typically specified by token issuers
1460 and security token services. This is typically a service-level statement but can be an endpoint-level
1461 statement.

1462 The schema for this optional element is shown below.

```
1463 <fed:ClaimDialectsOffered>  
1464   <fed:ClaimDialect Uri="xs:anyURI" /> +  
1465 </fed:ClaimDialectsOffered>
```

1466 The following describes the elements listed in the schema outlined above:

1467 `/fed:ClaimDialectsOffered`

1468 This element is used to express the list of claim dialects that the federating STS can understand
1469 and accept.

1470 `/fed:ClaimDialectsOffered/fed:ClaimDialect`

1471 This element indicates an individual claim dialect that the STS can understand.

1472 `/fed:ClaimDialectsOffered/fed:ClaimDialect/@Uri`

1473 This attribute provides the unique identifier (URI) of the individual claim dialect that the STS can
1474 understand.

1475 `/fed:ClaimDialectsOffered/fed:ClaimDialect/...`

1476 The semantics of any content for this element are undefined. Any extensibility or use of sub-
1477 elements MUST NOT alter the semantics defined in this specification.

1478 `/fed:ClaimDialectsOffered/fed:ClaimDialect/@{any}`

1479 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
1480 MUST NOT violate or alter the semantics defined in this specification.

1481 `/fed:ClaimDialectsOffered/@{any}`

1482 This extensibility mechanism allows attributes to be added. Use of this extensibility mechanism
1483 MUST NOT violate or alter the semantics defined in this specification.

1484 The following example illustrates using this optional element to specify that the issuing STS of the
1485 federating organization can accept the one standard claims dialect defined in this specification.

1486

```
1487 <fed:ClaimDialectsOffered>  
1488   <fed:ClaimDialect Uri="http://schemas.xmlsoap.org/ws/2005/05/fedclaims" />  
1489 </fed:ClaimDialectsOffered>
```

1490 3.1.12 AutomaticPseudonyms Element

1491 The OPTIONAL `<fed:AutomaticPseudonyms>` element allows a federation metadata provider to
1492 indicate if it automatically maps pseudonyms or applies some form of identity mapping. This element
1493 populates the [Federation Metadata] property. This is typically specified by token issuers and security
1494 token services. This is typically a service-level statement but can be an endpoint-level statement. If not
1495 specified, requestors SHOULD assume that the service does not perform automatic mapping (although it
1496 MAY).

1497 The schema for this optional element is shown below.

```
1498 <fed:AutomaticPseudonyms>  
1499   xs:boolean  
1500 </fed:AutomaticPseudonyms>
```

1501 **3.1.13 PassiveRequestorEndpoints Element**

1502 The optional `<fed:PassiveRequestorEndpoints>` element allows a federation metadata provider,
1503 security token service, or relying party to specify the endpoint address that supports the Web (Passive)
1504 Requestor protocol described below in section 13. This element populates the [Federation Metadata]
1505 property. This is an endpoint-level statement.

1506 The schema for this optional element is shown below.

```
1507 <fed:PassiveRequestorEndpoints>  
1508   <wsa:EndpointReference> ... </wsa:EndpointReference>+  
1509 </fed:PassiveRequestorEndpoints>
```

1510 The content of this element is an endpoint reference element as defined by [WS-Addressing] that
1511 identifies an endpoint address that supports receiving the Web (Passive) Requestor protocol messages
1512 described below in section 13.

1513 This element allows attributes to be added so long as they do not alter the semantics defined in this
1514 specification.

1515 It should be noted that this element MAY occur multiple times indicating distinct endpoints with different
1516 capabilities. Service providers MUST include functionally equivalent endpoints in a single
1517 `<fed:PassiveRequestorEndpoints>` element.

1518 The following example illustrates using this optional element to specify the endpoint address that supports
1519 the Web (Passive) Requestor protocol described in section 13 for the token issuing STS of the federating
1520 organization.

```
1521 <fed:PassiveRequestorEndpoints>  
1522   <wsa:EndpointReference>  
1523     <wsa:Address> http://fabrikam.com/federation/STS/Passive </wsa:Address>  
1524   </wsa:EndpointReference>  
1525 </fed:PassiveRequestorEndpoints>
```

1526

1527 **3.1.14 TargetScopes Element**

1528 The [WS-Trust] protocol allows a token requester to indicate the target where the issued token will be
1529 used (i.e., token scope) by using the optional element `wsp:AppliesTo` in the RST message. To
1530 communicate the supported `wsp:AppliesTo` (wtrealm values in passive requestor scenarios) for a realm,
1531 federated metadata provides the `<fed:TargetScopes>` element to indicate the EPRs that are associated
1532 with token scopes of the relying party or STS. Note that an RP or STS MAY be capable of supporting
1533 other `wsp:AppliesTo` values. This element populates the [Federation Metadata] property. This is typically
1534 a service-level statement.

1535 The schema for this optional element is shown below.

1536
1537
1538
1539
1540

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

1541 The following example illustrates using this optional element to specify a logical name of the federating
1542 organization as a token issuer.

1543
1544
1545
1546
1547

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

1548

1549 3.1.15 [Signature] Property

1550 The OPTIONAL [Signature] property provides a digital signature over the federation metadata document
1551 to ensure data integrity and provide data origin authentication. The provider of a federation metadata
1552 document SHOULD include a digital signature over the metadata document, and consumers of the
1553 metadata document SHOULD perform signature verification if a signature is present.

1554 The token used to sign this document MUST speak for the endpoint. If the metadata is for a token issuer
1555 then the key used to sign issued tokens SHOULD be used to sign this document. This means that if a
1556 <fed:TokenSigningKey> is specified, it SHOULD be used to sign this document.

1557 This section describes the use of [XML-Signature] to sign the federation metadata document, but other
1558 forms of digital signatures MAY be used for the [Signature] property. XML Signature is the
1559 RECOMMENDED signing mechanism. The [Signature] property (in the case of XML Signature this is
1560 represented by the <ds:Signature> element) provides the ability for a federation metadata provider
1561 organization to sign the metadata document such that a partner organization consuming the metadata
1562 can authenticate its origin.

1563 The signature over the federation metadata document MUST be signed using an enveloped signature
1564 format as defined by the [XML-Signature] specification. In such cases the root of the signature envelope
1565 MUST be the <fed:FederationMetadata> element as shown in the following example. If the
1566 metadata document is included inside another XML document, such as a SOAP message, the root of the
1567 signature envelope MUST remain the same. Additionally, XML Exclusive Canonicalization [XML-C14N]
1568 MUST be used when signing with [XML-Signature].

1569
1570
1571
1572
1573

```
(01) [<?xml version='1.0' encoding=... > ]
(02) <fed:FederationMetadata
(03)   xmlns:fed="..." xmlns:ds="..."
(04)   wsu:Id="_fedMetadata">
(05)   ...
```

```

1574 (06) <ds:Signature xmlns:ds="...">
1575 (07)   <ds:SignedInfo>
1576 (08)     <ds:CanonicalizationMethod Algorithm="..." />
1577 (09)     <ds:SignatureMethod Algorithm="..." />
1578 (10)     <ds:Reference URI="_fedMetadata">
1579 (11)       <ds:Transforms>
1580 (12)         <ds:Transform Algorithm=".../xmldsig#enveloped-signature" />
1581 (13)         <ds:Transform Algorithm=".../xml-exc-c14n#" />
1582 (14)       </ds:Transforms>
1583 (15)       <ds:DigestMethod Algorithm="..." />
1584 (16)       <ds:DigestValue>xdJRPBPERvaZD9gTt4e6Mg==</ds:DigestValue>
1585 (17)     </ds:Reference>
1586 (18)   </ds:SignedInfo>
1587 (19)   <ds:SignatureValue> mpcFEK6JuUFBPoJQ8VBW2Q==</ds:SignatureValue>
1588 (20)   <ds:KeyInfo>
1589 (21)     ...
1590 (22)   </ds:KeyInfo>
1591 (23) </ds:Signature>
1592 (24) </fed:FederationMetadata>

```

1593 Note that the enveloped signature contains a single `ds:Reference` element (line 10) containing a URI
1594 reference to the `<fed:FederationMetadata>` root element (line 04) of the metadata document.

1595

1596 3.1.16 Example Federation Metadata Document

1597 The following example illustrates a signed federation metadata document that uses the OPTIONAL
1598 metadata elements described above and an enveloped [XML Signature] to sign the document.

```

1599 <?xml version="1.0" encoding="utf-8" ?>
1600 <fed:FederationMetadata wsu:Id="_fedMetadata"
1601   xmlns:fed="..." xmlns:wsu="..." xmlns:wsse="..." xmlns:ds="..."
1602   xmlns:wsa="...">
1603   <fed:Federation>
1604     <fed:TokenSigningKeyInfo>
1605       <wsse:SecurityTokenReference>
1606         <ds:X509Data>
1607           <ds:X509Certificate>
1608             MIIBsTCCA+g...zRn3ZVIcvbQE=
1609           </ds:X509Certificate>
1610         </ds:X509Data>
1611       </wsse:SecurityTokenReference>
1612     </fed:TokenSigningKeyInfo>
1613     <fed:TokenIssuerName>
1614       http://fabrikam.com/federation/corporate
1615     </fed:TokenIssuerName>
1616     <fed:TokenIssuerEndpoint>
1617       <wsa:Address> http://fabrkam.com/federation/STS </wsa:Address>
1618     </fed:TokenIssuerEndpoint>
1619     <fed:TokenTypesOffered>
1620       <fed:TokenType Uri="urn:oasis:names:tc:SAML:1.1" />
1621       <fed:TokenType Uri="urn:oasis:names:tc:SAML:2.0" />
1622     </fed:TokenTypesOffered>
1623     <fed:ClaimTypesOffered>
1624       <auth:ClaimType Uri="http://.../claims/EmailAddr" >
1625         <auth:DisplayName>Email Address</auth:DisplayName>
1626       </auth:ClaimType>
1627       <auth:ClaimType Uri="http://.../claims/IsMember" >
1628         <auth:DisplayName>Is a Member (yes/no)</auth:DisplayName>
1629         <auth:Description>If a person is a member of this club</auth:Description>
1630       </auth:ClaimType>
1631     </fed:ClaimTypesOffered>

```

```

1632 </fed:ClaimTypesOffered> </fed:Federation>
1633
1634 <ds:Signature xmlns:ds="...">
1635   <ds:SignedInfo>
1636     <ds:CanonicalizationMethod Algorithm="..." />
1637     <ds:SignatureMethod Algorithm="..." />
1638     <ds:Reference URI="_fedMetadata">
1639       <ds:Transforms>
1640         <ds:Transform Algorithm=".../xmldsig#enveloped-signature" />
1641         <ds:Transform Algorithm=".../xml-exc-c14n#" />
1642       </ds:Transforms>
1643       <ds:DigestMethod Algorithm="..." />
1644       <ds:DigestValue>xdJRPBPERvaZD9gTt4e6Mg==</ds:DigestValue>
1645     </ds:Reference>
1646   </ds:SignedInfo>
1647   <ds:SignatureValue>mpcFEK6JuUFBPojQ8VBW2Q==</ds:SignatureValue>
1648   <ds:KeyInfo>
1649     ...
1650   </ds:KeyInfo>
1651 </ds:Signature>
1652 </fed:FederationMetadata>

```

1653 3.2 Acquiring the Federation Metadata Document

1654 This section provides specific details and restrictions on how a party may securely obtain the federation
 1655 metadata document for a *target domain* representing a target organization it wishes to federate with. It
 1656 should be noted that some providers of federation metadata documents MAY require authentication of
 1657 requestors or MAY provide different (subset) documents if requestors are not authenticated.

1658 It is assumed that the target domain is expressed as a fully-qualified domain name (FQDN). In other
 1659 words, it is expressed as the DNS domain name of the target organization, e.g., fabrikam.com.

1660 It should be noted that compliant services are NOT REQUIRED to support all of the mechanisms defined
 1661 in this section. If a client only has a DNS host name and wants to obtain the federation metadata, the
 1662 following order is the RECOMMENDED bootstrap search order:

- 1663 1. Use the well-known HTTPS address with the federation ID
- 1664 2. Use the well-known HTTPS address for the default federation
- 1665 3. Use the well-known HTTP address with the federation ID
- 1666 4. Use the well-known HTTP address for the default federation
- 1667 5. Look for any DNS SRV records indicating federation metadata locations

1668 If multiple locations are available and no additional prioritization is specified, the following order is the
 1669 RECOMMENDED download processing order:

- 1670 1. HTTPS
- 1671 2. WS-Transfer/WS-ResourceTransfer
- 1672 3. HTTP

1673 3.2.1 WSDL

1674 The metadata document MAY be included within a WSDL document using the extensibility mechanisms
 1675 of WSDL. Specifically the `<fed:FederationMetadata>` element can be placed inside of WSDL
 1676 documents in the same manner as policy documents are as specified in WS-PolicyAttachment.

1677 The metadata document can appear in WSDL for a service, port, or binding.

1678 3.2.2 The Federation Metadata Path

1679 A default path MAY be supported to provide federation metadata. The path for obtaining the federation
1680 metadata document for the default federation for a target domain denoted by **target-DNS-domain**
1681 SHOULD be constructed as follows:

1682 `http://server-name/FederationMetadata/spec-version/FederationMetadata.xml`

1683 or

1684 `https://server-name/FederationMetadata/spec-version/FederationMetadata.xml`

1685 where

1686 *server-name* is the host name (DNS name) of a server providing the federation metadata document. It
1687 SHOULD be obtained by doing a DNS query of SRV records for **target-DNS-domain** as
1688 described in Section 3.2.6. If no DNS record is found, then the target DNS domain name MUST
1689 BE used as the default value of the server name as well.

1690 *spec-version* is the version of the federation metadata specification supported by the acquiring party. For
1691 this version of the specification the **spec-version** MUST BE the string "2007-06".

1692 Implementations MAY choose to use a short form of the target DNS domain name, such as the primary
1693 domain and suffix, but this choice is implementation specific.

1694 The following subsections describe the mechanisms through which the federation metadata document for
1695 a target domain may be acquired by a federating party. The target domain MUST support at least one of
1696 the mechanisms described below, but MAY choose to support more than one mechanism.

1697 It is RECOMMENDED that a target domain (or organization) that makes federation metadata available for
1698 acquisition by partners SHOULD publish DNS SRV resource records to allow an acquiring party to locate
1699 the servers where the metadata is available. The type and format of the SRV resource records to be
1700 published in DNS is described in Section 3.2.6. These records correspond to each metadata acquisition
1701 mechanism specified in the following subsections.

1702 If a specific federation context is known, the following URLs SHOULD be checked prior to checking for
1703 the default federation context.

1704 `http://server-name/FederationMetadata/spec-version/fed-id/FederationMetadata.xml`

1705 or

1706 `https://server-name/FederationMetadata/spec-version/fed-id/FederationMetadata.xml`

1707 where

1708 *fed-id* is the `FederationID` value described previously for identifying a specific federation.

1709 3.2.3 Retrieval Mechanisms

1710 The following OPTIONAL retrieval mechanisms are defined:

1711 Using HTTP

1712 The federation metadata document may be obtained from the following URL using HTTP GET
1713 mechanism:

1714 `http: path`

1715 where *path* is constructed as described in Section 3.2.2.

1716 Metadata signatures are RECOMMENDED when using HTTP download.

1717 Using HTTPS

1718 The federation metadata document MAY be obtained from the following URL using HTTPS GET
1719 mechanism:

1720

```
https:path
```

1721 where *path* is constructed as described in Section 3.2.2.

1722 There is no requirement that the HTTPS server key be related to the signing key identified in the
1723 metadata document, but it is RECOMMENDED that requestors verify that both keys can speak for the
1724 target service.

1725 Using WS-Transfer/WS-ResourceTransfer

1726 The federation metadata document can be obtained by sending the [WS-Transfer] "Get" operation to an
1727 endpoint that serves that metadata as described in [WS-MetadataExchange] (see also section 3.2.5).
1728 Note that the [WS-ResourceTransfer] extensions MAY be used to filter the metadata information returned.
1729 The use of [WS-Security] with [WS-Transfer/WS-ResourceTransfer] is RECOMMENDED to authenticate
1730 the sender and protect the integrity of the message.

1731 3.2.4 FederatedMetadataHandler Header

1732 If an endpoint reference for metadata obtained via SOAP requests is not already available to a requester
1733 (e.g. when only a URL is know), the requestor SHOULD include the

1734 <fed:FederationMetadataHandler> header to allow metadata requests to be quickly identified.

1735 The syntax is as follows:

```
1736 <fed:FederationMetadataHandler .../>
```

1737 The<fed:FederationMetadataHandler> header SHOULD NOT use a S:mustUnderstand='1'
1738 attribute. Inclusion of this header allows a front-end service to know that federation metadata is being
1739 requested and perform header-based routing.

1740 The following example illustrates a [WS-Transfer] with [WS-ResourceTransfer] extensions request
1741 message to obtain the federation metadata document for an organization with contoso.com as its domain
1742 name.

```
1743 (01) <s12:Envelope  
1744 (02)   xmlns:s12="..."  
1745 (03)   xmlns:wsa="..."  
1746 (04)   xmlns:wsxf="..."  
1747 (05)   xmlns:fed="...">  
1748 (06)   <s12:Header>  
1749 (07)     <wsa:Action>  
1750 (08)       http://schemas.xmlsoap.org/ws/2004/09/transfer/Get  
1751 (09)     </wsa:Action>  
1752 (10)     <wsa:MessageID>  
1753 (11)       uuid:73d7edfd-5c3d-b949-46ba-02decaee433f  
1754 (12)     </wsa:MessageID>  
1755 (13)     <wsa:ReplyTo>  
1756 (14)       <wsa:Address>http://fabrikam.com/Endpoint</wsa:Address>  
1757 (15)     </wsa:ReplyTo>  
1758 (16)     <wsa:To>  
1759 (17)       http://contoso.com/FederationMetadata/2007-06/FederationMetadata.xml  
1760 (18)     </wsa:To>  
1761 (19)     <fed:FederatedMetadataHandler />  
1762 (20)   </s12:Header>  
1763 (21)   <s12:Body />  
1764 (22) </s12:Envelope>
```

1765 The response to the [WS-Transfer] with [WS-ResourceTransfer] extensions request message is illustrated
1766 below.

```
1767 (01) <s12:Envelope  
1768 (02)   xmlns:s12="..."
```

```

1769 (03)   xmlns:wsa="..."
1770 (04)   xmlns:wsxf="..."
1771 (05)   xmlns:fed="...">
1772 (06)   <s12:Header>
1773 (07)     <wsa:To>http://fabrikam.com/Endpoint</wsa:To>
1774 (08)     <wsa:Action>
1775 (09)       http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
1776 (10)     </wsa:Action>
1777 (11)     <wsa:MessageID>
1778 (12)       uuid:86d7eac5-6e3d-b869-64bc-35edacee743d
1779 (13)     </wsa:MessageID>
1780 (14)     <wsa:RelatesTo>
1781 (15)       uuid:73d7edfd-5c3d-b949-46ba-02decaee433f
1782 (16)     </wsa:RelatesTo>
1783 (17)   </s12:Header>
1784 (18)   <s12:Body>
1785 (19)     <fed:FederationMetadata
1786 (20)       xmlns:fed="...">
1787 (21)       ...
1788 (22)     </fed:FederationMetadata>
1789 (21) </s12:Body>
1790 (22) </s12:Envelope>

```

1791 3.2.5 Metadata Exchange Dialect

1792 The federation metadata document MAY be included as a metadata unit within a Web service
1793 <mex:Metadata> element, which is a collection of metadata units, using the metadata unit inclusion
1794 mechanisms described in [WS-MetadataExchange]. This can be done by including a
1795 <mex:MetadataSection> element that contains the federation metadata document in-line or by
1796 reference. To facilitate inclusion of the federation metadata as a particular type of metadata unit, the
1797 following metadata dialect URI is defined in this specification that MUST be used as the value of the
1798 <mex:MetadataSection/@Dialect> XML attribute:

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

1800 No identifiers for federation metadata units, as specified by the value of the OPTIONAL
1801 <mex:MetadataSection/@Identifier> XML attribute, are defined in this specification.

1802 For example, a federation metadata unit specified in-line within a <mex:Metadata> element is shown
1803 below:

```

1804 <mex:Metadata>
1805   <mex:MetadataSection
1806     Dialect='http://docs.oasis-open.org/wsfed/federation/200706'>
1807     <fed:FederationMetadata ...>
1808     ...
1809   </fed:FederationMetadata>
1810   <mex:MetadataSection>
1811 </mex:Metadata>

```

1812 3.2.6 Publishing Federation Metadata Location

1813 A target domain (or organization) that makes federation metadata available for acquisition by partners
1814 SHOULD publish SRV resource records in the DNS database to allow an acquiring party to locate the

1815 servers where the metadata is available. The specific format and content of the SRV resource records to
1816 be published is described here.

1817 The SRV record is used to map the name of a service (in this case the federation metadata service) to
1818 the DNS hostname of a server that offers the service. For more information about SRV resource records,
1819 see [DNS-SRV-RR]. The general form of the *owner name* of a SRV record to be published is as follows:

1820 `_Service.Protocol.TargetDnsDomain`

1821 In this case, a target domain offers the “federation metadata” service over one or more of the protocol
1822 mechanisms described earlier (namely, HTTP, HTTPS or WS-Transfer/WS-ResourceTransfer). For each
1823 protocol mechanism supported by a target domain, a corresponding SRV record SHOULD published in
1824 DNS as follows.

1825 If acquisition of the federation metadata document using HTTP GET (Section 3.2.3) is supported, then the
1826 owner name of the published SRV record MUST be of the form below:

1827 `_fedMetadata._http.TargetDnsDomain`

1828 If acquisition of the federation metadata document using HTTPS GET (Section 3.2.3) is supported, then
1829 the owner name of the published SRV record MUST be of the form below:

1830 `_fedMetadata._https.TargetDnsDomain`

1831 If acquisition of the federation metadata document using [WS-Transfer/WS-ResourceTransfer] (Section
1832 3.2.3) is supported, then the owner name of the published SRV record MUST be of the form below:

1833 `_fedMetadata._wsxfr._http.TargetDnsDomain`

1834 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.

1835 Note that if multiple protocols are specified with the same priority, the requestor MAY use any protocol or
1836 process in any order it chooses.

1837 The following example illustrates the complete SRV records published by the organization with domain
1838 name “contoso.com” that makes its federation metadata available over all three mechanisms discussed
1839 earlier.

1840

```
1841 server1.contoso.com IN A 128.128.128.0  
1842 server2.contoso.com IN A 128.128.128.1  
1843 _fedMetadata._http.contoso.com IN SRV 0 0 80 server1.contoso.com  
1844 _fedMetadata._https.contoso.com IN SRV 0 0 443 server1.contoso.com  
1845 _fedMetadata._wsxfr.contoso.com IN SRV 0 0 80 server2.contoso.com
```

1846 A client attempting to acquire the federation metadata for a target domain using any selected protocol
1847 mechanism SHOULD query DNS for SRV records using one of the appropriate owner name forms
1848 described above.

1849 **3.2.7 Federation Metadata Acquisition Security**

1850 It is RECOMMENDED that a target domain publishing federation metadata SHOULD include a signature
1851 in the metadata document using a key that is authorized to "speak for" the target domain. If the metadata
1852 contains a `<fed:TokenSigningKey>` element then this key SHOULD be used for the signature. If
1853 there are multiple `Federation` elements specified then the default scope's signing key SHOULD be
1854 used. If there is no default scope then the choice is up to the signer. Recipients of federation metadata
1855 SHOULD validate that signature to authenticate the metadata publisher and verify the integrity of the
1856 data. Specifically, a recipient SHOULD verify that the key used to sign the document has the right to
1857 "speak for" the target domain (see *target-DNS-domain* in Section 3.2.2) with which the recipient is trying
1858 to federate. See also the security considerations at the end of this document.

4 Sign-Out

1859

1860 The purpose of a *federated sign-out* is to clean up any cached state and security tokens that may exist
1861 within the federation, but which are no longer required. In typical usage, sign-out notification serves as a
1862 hint – upon termination of a principal's session – that it is OK to flush cached data (such as security
1863 tokens) or state information for that specific principal. It should be noted that a sign-out message is a
1864 *one-way* message. No "sign-out-complete" reply message can be required since the Sign-Out operation
1865 cannot be guaranteed to complete. Further, sign-out requests might be processed in batch, causing a
1866 time delay that is too long for the request and response to be meaningfully correlated. In addition,
1867 requiring a Web browser requestor to wait for a successful completion response could introduce arbitrary
1868 and lengthy delays in the user experience. The processing implication of sign-out messages can vary
1869 depending on the type of application that is being used to sign-out. For example, the implication of sign-
1870 out on currently active transactions is undefined and is resource-specific.

1871 In some cases, formal sign-out is implicit or not required. This section defines messages that MAY be
1872 used by profiles for explicit sign-out.

1873 In general, sign-out messages are unreliable and correct operation must be ensured in their absence (i.e.,
1874 the messages serve as hints only). Consequently, these messages MUST also be treated as idempotent
1875 since multiple deliveries could occur.

1876 When sign-out is supported, it is typically provided as part of the IP/STS as it is usually the central
1877 processing point.

1878 Sign-out is separate from token cancellation as it applies to all tokens and all target sites for the principal
1879 within the domain/realm.

4.1 Sign-Out Message

1881 The sign-out mechanism allows requestors to send a message to its IP/STS indicating that the requester
1882 is initiating a termination of the SSO. That is, cached information or state information can safely be
1883 flushed. This specification defines OPTIONAL sign-out messages that MAY be used. It should be noted,
1884 however, that the typical usage pattern is that only token issuance and message security are used and
1885 sign-out messages are only for special scenarios. Sign-out messages, whether from the client to the
1886 IP/STS, from the IP/STS to a subscriber, or from the client to a service provider, all use the same
1887 message form described in this section.

1888 For SOAP, the action of this message is as follows:

1889 `http://docs.oasis-open.org/wsfed/federation/200706/SignOut`

1890 The following represents an overview of the syntax of the `<fed:SignOut>` element:

```
1891 <fed:SignOut wsu:Id="..." ...>  
1892   <fed:Realm>xs:anyURI</fed:Realm> ?  
1893   <fed:SignOutBasis ...>...<fed:SignOutBasis>  
1894   ...  
1895 </fed:SignOut>
```

1896 The following describes elements and attributes used in a `<fed:SignOut>` element.

1897 `/fed:SignOut`

1898 This element represents a sign-out message.

1899 `/fed:SignOut/fed:Realm`

1900 This OPTIONAL element specifies the "realm" to which the sign-out applies and is specified as a
1901 URI. If no realm is specified, then it is assumed that the recipient understands and uses a
1902 fixed/default realm.

1903 /fed:SignOut/fed:SignOutBasis

1904 The contents of this REQUIRED element indicate the principal that is signing out. Note that any
 1905 security token or security token reference MAY be used here and multiple tokens MAY be
 1906 specified. That said, it is expected that the <UsernameToken> will be the most common. Note
 1907 that a security token or security token reference MUST be specified.

1908 /fed:SignOut/fed:SignOutBasis/@{any}

1909 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added
 1910 to the element. Use of this extensibility mechanism MUST NOT alter the semantics of this
 1911 specification.

1912 /fed:SignOut/fed:SignOutBasis/{any}

1913 This is an extensibility mechanism to allow the inclusion of the relevant security token reference
 1914 or security token(s).

1915 /fed:SignOut/@wsu:Id

1916 This OPTIONAL attribute specifies a string label for this element.

1917 /fed:SignOut/@{any}

1918 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added
 1919 to the element. Use of this extensibility mechanism MUST NOT alter the semantics of this
 1920 specification.

1921 /fed:SignOut/{any}

1922 This is an extensibility mechanism to allow additional elements to be used. For example, an STS
 1923 might use extensibility to further qualify the sign-out basis. Use of this extensibility mechanism
 1924 MUST NOT alter the semantics of this specification.

1925

1926 The <fed:SignOut> message SHOULD be signed by the requestor to prevent tampering and to
 1927 prevent unauthorized sign-out messages (i.e., Alice sending a sign-out message for Bob without Bob's
 1928 knowledge or permission). The signature SHOULD contain a timestamp to prevent replay attacks (see
 1929 WS-Security for further discussion on this). It should be noted, however, that a principal MAY delegate
 1930 the right to issue such messages on their behalf. The following represents an example of the
 1931 <fed:SignOut> message:

```

1932 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
1933   xmlns:wsu="..." xmlns:wsse="...">
1934   <S:Header>
1935     ...
1936     <wsu:Timestamp wsu:Id="ts">
1937       ...
1938     </wsu:Timestamp>
1939     <wsse:Security>
1940       <!-- Signature referecing IDs "ts" & "so" -->
1941       ...
1942     </wsse:Security>
1943   </S:Header>
1944   <S:Body>
1945     <fed:SignOut wsu:Id="so">
1946       <fed:SignOutBasis>
1947         <wsse:UsernameToken>
1948           <wsse:Username>NNK</wsse:Username>
1949         </wsse:UsernameToken>
1950       </fed:SignOutBasis>
1951     </fed:SignOut>
1952   </S:Body>
1953 </S:Envelope>

```

1954 4.2 Federating Sign-Out Messages

1955 In many environments there is a need to take the messages indicating sign-out and distribute them
1956 across the federation, subject to authorization and privacy rules. Consequently, these messages result
1957 from when an explicit message is sent to the IP/STS (by either the principal or a delegate such as an
1958 IP/STS), or implicitly from an IP/STS as a result of some other action (such as a token request).

1959 In the typical use case, federated sign-out messages will be generated by the principal terminating a
1960 session, either at the “primary STS” (the IP/STS that manages the principal’s identity) or at one of the
1961 resource providers (or its STS) accessed during the session. There are two primary flows for these
1962 messages. In one case they are effectively chained through all the STSs involved in the session; that is,
1963 a mechanism is used (if available) by the “primary STS” to send sign-out messages to all the other STSs
1964 in a sequential manner by causing each message to cause the next message to occur in sequence
1965 resulting in a message back to itself either on completion or at each step to orchestrate the process. The
1966 second approach is to require the “primary STS” to send sign-out messages to all the other token
1967 services and target services in parallel (those that it knows about).

1968 The chained (sequential) approach has been found to be fragile. If one of the message fails to complete
1969 its local processing and does not pass the sign-out message on – or the network partitions – the sign-out
1970 notification does not reach all the involved parties. For this reason, compliant implementations SHOULD
1971 employ the parallel approach. If the session is terminated at a resource provider, it SHOULD clean up
1972 any local state and then send a sign-out message to the “primary STS”. The latter SHOULD send parallel
1973 sign-out messages to all the other STSs.

1974 Sessions MAY involve secondary branches (between token services at different resources) of which the
1975 “primary STS” has no knowledge. In these cases, the appropriate resource token services SHOULD
1976 perform the role of “primary STS” for sign-out of these branches.

1977 It should be noted that clients MAY also push (send) sign-out messages directly to other services such as
1978 secondary IP/STSs or service providers.

1979 Sign-out could potentially be applied to one of two different scopes for the principal’s session. Sign-out
1980 initiated at the “primary STS” SHOULD have global scope and apply to all resource STSs and all
1981 branches of the session. Sign-out initiated at a resource STS could also have global scope as described
1982 above. However, it could also be considered as a request to clean up only the session state related to
1983 that particular resource provider. Thus implementations MAY provide a mechanism to restrict the scope
1984 of federated sign-out requests that originate at a resource STS to its particular branch of the principal’s
1985 session. This SHOULD result in cleaning up all state at (or centered upon) that STS. It SHOULD involve
1986 a request to be sent to the “primary STS” to clean up session state only for that particular STS or
1987 resource provider.

1988 Federated sign-out request processing could involve providing status messages to the user. This
1989 behavior is implementation specific and out-of-scope of this specification.

1990 The result of a successful request is that all compliant SSO messages generated implicitly or explicitly are
1991 sent to the requesting endpoints if allowed by the authorization/privacy rules.

1992 SSO messages MAY be obtained by subscribing to the subscription endpoint using the mechanisms
1993 described in [WS-Eventing]. The subscription endpoint, if available, is described in the federation
1994 metadata document.

1995 The [WS-Eventing] mechanisms allow for subscriptions to be created, renewed, and cancelled. SSO
1996 subscriptions MAY be filtered using the XPath filter defined in [WS-Eventing] or using the SSO filter
1997 specified by the following URI:

1998 `http://docs.oasis-open.org/wsfed/federation/200706/ssoevt`

1999 This filter allows the specification of a realm and security tokens to restrict the SSO messages. The
2000 syntax is as follows:

2001
2002
2003
2004
2005
2006
2007
2008

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

2009 The following describes elements and attributes illustrated above:

2010 /wse:Filter/fed:Realm

2011 This OPTIONAL element specifies the "realm" to which the sign-out applies. At most one
2012 <fed:Realm> can be specified. The contents of this element are the same type and usage as in
2013 the *fed:Signout/fed:Realm* described above. If this element is not specified it is assumed
2014 that either the subscription service knows how to infer the correct realm and uses a single
2015 service-determined realm or the request fails. Note that if multiple realms are desired then
2016 multiple subscriptions are needed.

2017 /wse:Filter/... security tokens(s) ...

2018 The contents of these OPTIONAL elements restrict messages to only the specified identities.
2019 Note that any security token or security token reference MAY be used here and multiple tokens
2020 MAY be specified. That said, it is expected that the *<wsse:UsernameToken>* will be the most
2021 common. Note that if multiple tokens are specified they represent a logical OR – that is,
2022 messages that match any of the tokens for the corresponding realm are reported.

2023 This filter dialect does not allow any contents other than those described above. If no filter is specified
2024 then the subscription service MAY fail or MAY choose a default filter for the subscription.

2025

5 Attribute Service

2026

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

2032

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

2034

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

2035

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

2036

2037

2038

2039

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

2040

2041

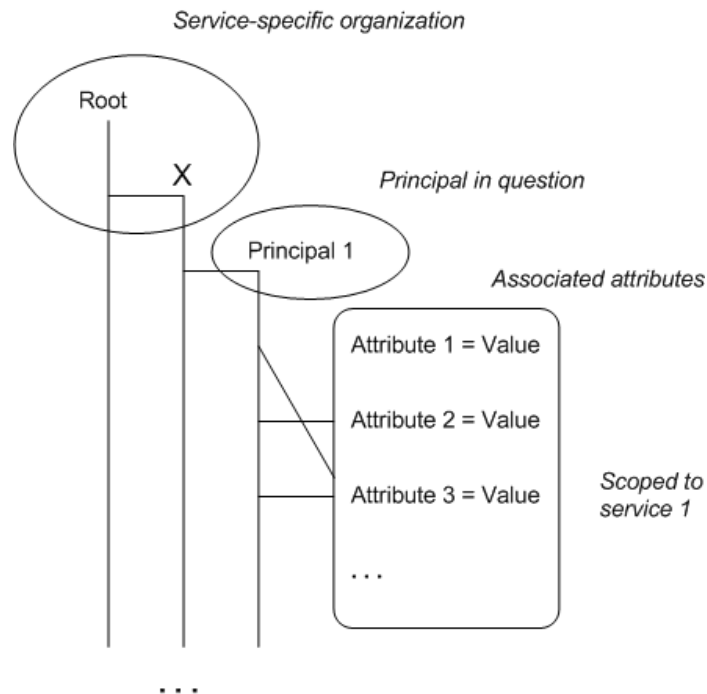
2042

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

2043

2044

2045



2046

2047

Figure 21 Attribute Service

2048

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

2049

2050

2051

2052 Different services MAY support different types of attribute services which MAY be identified via policy by
2053 definition of new policy assertions indicating the attribute service supported.

2054 Each attribute store MAY support different subsets of the functionality as described above. The store's
2055 policy indicates what functionality it supports.

2056 This specification does not require a specific attribute service definition or interface. However, as
2057 indicated in sections 2.7 and 3.1.8, the WS-Trust Security Token Service interface and token issuance
2058 protocol MAY be used as the interface to an attribute service. Reusing an established service model and
2059 protocol could simplify threat analysis and implementation of attribute services.

2060

6 Pseudonym Service

2061

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

2062

2063

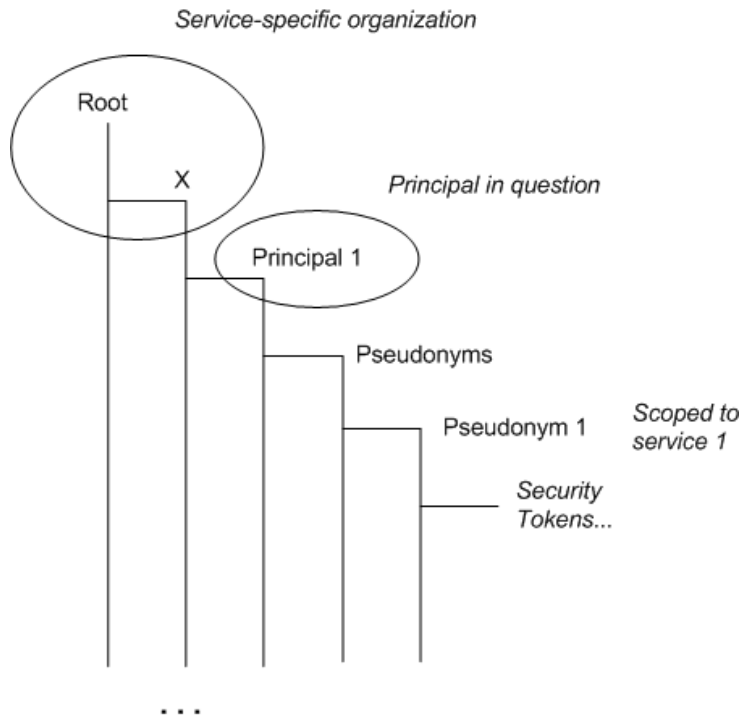
Pseudonym services MAY exist in some form already in service environments using service-specific protocols. This specification defines an additional, generic, interface to these services for interoperability with Web services.

2064

2065

2066

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



2067

2068

Figure 22 Pseudonym Service

2069

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.

2070

2071

2072

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).

2073

2074

2075

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.

2076

2077

2078

2079

2080

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).

2081

2082

2083

2084

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

2085

2086 6.1 Filtering Pseudonyms

2087 When performing operations on a pseudonym store it is RECOMMENDED to filter the scope of the
2088 operation. This is done using the following dialect with the [WS-ResourceTransfer] extensions to [WS-
2089 Transfer]:

2090 `http://docs.oasis-open.org/wsfed/federation/200706/pseudonymdialect`

2091 Alternatively, the <fed:FilterPseudonyms> header MAY be specified with WS-Transfer to allow
2092 filtering to be specified as part of an endpoint reference (EPR).

2093 The syntax for the <fed:FilterPseudonyms> element is as follows:

```
2094 <fed:FilterPseudonyms ...>  
2095   <fed:PseudonymBasis ...>...</fed:PseudonymBasis> ?  
2096   <fed:RelativeTo ...>...</fed:RelativeTo> ?  
2097   ...  
2098 </fed:FilterPseudonyms>
```

2099 The following describes elements and attributes used in a <fed:FilterPseudonyms> element.

2100 /fed:FilterPseudonyms

2101 This element indicates a request to filter a pseudonym operation based on given identity
2102 information and applicability scope.

2103 /fed:FilterPseudonyms/fed:PseudonymBasis

2104 This element specifies a security token or security token reference identifying the known identity
2105 information. This element is typically required to identify the basis but MAY be omitted if the
2106 context is known. This specification places no requirements on what information (claims) are
2107 required to be a pseudonym basis – that can vary by service.

2108 /fed:FilterPseudonyms/fed:PseudonymBasis/@{any}

2109 This is an extensibility point allowing attributes to be specified. Use of this extensibility
2110 mechanism MUST NOT alter semantics defined in this specification.

2111 /fed:FilterPseudonyms/fed:PseudonymBasis/{any}

2112 This is an extensibility mechanism to allow the inclusion of the relevant security token reference
2113 or security token.

2114 /fed:FilterPseudonyms/fed:RelativeTo

2115 This RECOMMENDED element indicates the scope for which the pseudonym is requested. This
2116 element has the same type as <wsp:AppliesTo>.

2117 /fed:FilterPseudonyms/fed:RelativeTo/@{any}

2118 This is an extensibility point allowing attributes to be specified.

2119 Use of this extensibility mechanism MUST NOT alter the semantics of this specification.

2120 alter semantics defined in this specification.

2121 /fed:FilterPseudonyms/@{any}

2122 This is an extensibility point allowing attributes to be specified. Use of this extensibility
2123 mechanism MUST NOT . alter semantics defined in this specification.

2124 /fed:FilterPseudonyms/{any}

2125 This is an extensibility point allowing content elements to be specified.

2126 Use of this extensibility mechanism MUST NOT alter semantics defined in this specification.

2127 As noted above, in some circumstances it MAY be desirable to include a filter as part of an EPR. To
2128 accommodate this, <fed:FilterPseudonyms> element MAY be specified as a SOAP header. It is
2129 RECOMMENDED that the SOAP *mustUnderstand* attribute be specified as *true* whenever this is used as
2130 a header. If a <fed:FilterPseudonyms> header is specified, the message MUST NOT contain
2131 additional filtering.

2132 6.2 Getting Pseudonyms

2133 Pseudonyms are requested from a pseudonym service using the [WS-Transfer] “GET” method with the
2134 [WS-ResourceTransfer] extensions. The dialect defined in 6.1 (or the <fed:FilterPseudonyms>
2135 header) is used to restrict the pseudonyms that are returned.

2136 Pseudonyms are returned in the body of the GET response message in a <fed:Pseudonym> element
2137 as follows:

```
2138 <fed:Pseudonym ...>  
2139   <fed:PseudonymBasis ...>...</fed:PseudonymBasis>  
2140   <fed:RelativeTo ...>...</fed:RelativeTo>  
2141   <wsu:Expires>...</wsu:Expires>  
2142   <fed:SecurityToken ...>...</fed:SecurityToken> *  
2143   <fed:ProofToken ...>...</fed:ProofToken> *  
2144   ...  
2145 </fed:Pseudonym>
```

2146 The following describes elements and attributes described above:

2147 /fed:Pseudonym

2148 This element represents a pseudonym for a principal.

2149 /fed:Pseudonym/fed:PseudonymBasis

2150 This element specifies a security token or security token reference identifying the known identity
2151 information (see [WS-Security]). Often this is equivalent to the basis in the request although if
2152 multiple pseudonyms are returned that value may be different.

2153 /fed:Pseudonym/fed:PseudonymBasis/@{any}

2154 This is an extensibility point allowing attributes to be specified.

2155 Use of this extensibility mechanism MUST NOT alter semantics defined in this specification.

2156 /fed:Pseudonym/fed:PseudonymBasis/{any}

2157 This is an extensibility mechanism to allow the inclusion of the relevant security token reference
2158 or security token. Use of this extensibility mechanism MUST NOT alter semantics defined in this
2159 specification.

2160 /fed:Pseudonym/fed:RelativeTo

2161 This REQUIRED element indicates the scope for which the pseudonym is requested. This
2162 element has the same type as <wsp:AppliesTo>.

2163 /fed:Pseudonym/fed:RelativeTo/@{any}

2164 This is an extensibility point allowing attributes to be specified. Use of this extensibility
2165 mechanism MUST NOT alter semantics defined in this specification.

2166 /fed:Pseudonym/wsu:Expires

2167 This OPTIONAL element indicates the expiration of the pseudonym.

2168 /fed:Pseudonym/fed:SecurityToken

2169 This OPTIONAL element indicates a security token for the scope. Note that multiple tokens MAY
2170 be specified.

2171 /fed:Pseudonym/fed:SecurityToken/@{any}
 2172 This is an extensibility point allowing attributes to be specified. Use of this extensibility
 2173 mechanism MUST NOT alter semantic defined in this specification.

2174 /fed:Pseudonym/fed:SecurityToken/{any}
 2175 This is an extensibility mechanism to allow the inclusion of the relevant security token(s). Use of
 2176 this extensibility mechanism MUST NOT alter semantics defined in this specification

2177 /fed:Pseudonym/fed:ProofToken
 2178 This OPTIONAL element indicates a proof token for the scope. Note that multiple tokens MAY be
 2179 specified.

2180 /fed:Pseudonym/fed:ProofToken/@{any}
 2181 This is an extensibility point allowing attributes to be specified. Use of this extensibility
 2182 mechanism MUST NOT alter semantics defined in this specification.

2183 /fed:Pseudonym/fed:ProofToken/{any}
 2184 This is an extensibility mechanism to allow the inclusion of the relevant security token(s). Use of
 2185 this extensibility mechanism MUST NOT alter semantics defined in this specification.

2186 /fed:Pseudonym/@{any}
 2187 This is an extensibility point allowing attributes to be specified. Use of this extensibility
 2188 mechanism MUST NOT alter semantics defined in this specification.

2189 /fed:Pseudonym/{any}
 2190 This is an extensibility point allowing content elements to be specified. Use of this extensibility
 2191 mechanism MUST NOT alter semantics defined in this specification.

2192 For example, the following example obtains the local pseudonym associated with the identity (indicated
 2193 binary security token) for the locality (target scope) indicated by the URI
 2194 <http://www.fabrikam123.com/NNK>.

```

2195 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
2196   xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
2197   <S:Body>
2198     <wsrt:Get
2199       Dialect="http://docs.oasis-open.org/wsrfed/federation/200706/pseudonymdialect">
2200       <wsrt:Expression>
2201         <fed:FilterPseudonyms>
2202           <fed:PseudonymBasis>
2203             <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2204           </fed:PseudonymBasis>
2205           <fed:RelativeTo>
2206             <wsa:Address>
2207               http://www.fabrikam123.com/NNK
2208             </wsa:Address>
2209           </fed:RelativeTo>
2210         </fed:FilterPseudonyms>
2211       </wsrt:Expression>
2212     </wsrt:Get>
2213   </S:Body>
2214 </S:Envelope>
  
```

2215 A sample response might be as follows:

```

2216 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
2217   xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
2218   <S:Body>
2219     <wsrt:GetResponse>
2220     <wsrt:Result>
  
```

```

2221     <fed:Pseudonym>
2222         <fed:RelativeTo>
2223             <wsa:Address>
2224                 http://www.fabrikam123.com/NNK
2225             </wsa:Address>
2226         </fed:RelativeTo>
2227         <wsu:Expires>2003-12-10T09:00Z</wsu:Expires>
2228         <fed:SecurityToken>...</fed:SecurityToken>
2229         <fed:ProofToken>...</fed:ProofToken>
2230     </fed:Pseudonym>
2231 </wsrt:Result>
2232 </wsrt:GetResponse>
2233 </S:Body>
2234 </S:Envelope>

```

2235 6.3 Setting Pseudonyms

2236 Pseudonyms are updated in a pseudonym service using the [WS-Transfer] “PUT” operation with the [WS-
2237 ResourceTransfer] extensions using the dialect defined in 6.1 (or the <fed:FilterPseudonyms>
2238 header). This allows one or more pseudonyms to be added. If a filter is not specified, then the PUT
2239 impacts the full pseudonym set. It is RECOMMENDED that filters be used.

2240 The following example sets pseudonym associated with the identity (indicated binary security token) for
2241 the locality (target scope) indicated by the URI http://www.fabrikam123.com/NNK.

```

2242 <S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:wsxf="..." xmlns:fed="..."
2243     xmlns:wsu="..." xmlns:wsse="..." xmlns:wsrt="...">
2244 <S:Body>
2245 <wsrt:Put
2246     Dialect="http://docs.oasis-open.org/wsrfed/federation/200706/pseudonymdialect">
2247 <wsrt:Fragment Mode="Inset">
2248 <wsrt:Expression>
2249 <fed:FilterPseudonyms>
2250 <fed:PseudonymBasis>
2251 <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2252 </fed:PseudonymBasis>
2253 <fed:RelativeTo>
2254 <wsa:Address>
2255     http://www.fabrikam123.com/NNK
2256 </wsa:Address>
2257 </fed:RelativeTo>
2258 </fed:FilterPseudonyms>
2259 </wsrt:Expression>
2260 <wsrt:Value>
2261 <fed:Pseudonym>
2262 <fed:PseudonymBasis>
2263 <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2264 </fed:PseudonymBasis>
2265 <fed:RelativeTo>
2266 <wsa:Address>
2267     http://www.fabrikam123.com/NNK
2268 </wsa:Address>
2269 </fed:RelativeTo>
2270 <fed:SecurityToken>
2271 <wsse:UsernameToken>
2272 <wsse:Username> "Nick" </wsse:Username>
2273 </wsse:UsernameToken>
2274 </fed:SecurityToken>
2275 <fed:ProofToken>...</fed:ProofToken>
2276 </fed:Pseudonym>
2277 </wsrt:Value>
2278 </wsrt:Fragment>

```

2279
2280
2281

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

2282

6.4 Deleting Pseudonyms

2283 Pseudonyms are deleted in a pseudonym service using the [WS-Transfer] “PUT” operation with the [WS-
2284 ResourceTransfer] extensions. The dialect defined in 6.1 (or the <fed:FilterPseudonyms> header) is
2285 used to restrict the scope of the “PUT” to only remove pseudonym information corresponding to the filter.
2286 If a filter is not specified, then the PUT impacts the full pseudonym set. It is RECOMMENDED that filters
2287 be used.

2288 The following example deletes the pseudonym associated with the identity (indicated binary security
2289 token) for the locality (target scope) indicated by the URI <http://www.fabrikam123.com/NNK>.

2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311

```
<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/wsrfed/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>
```

2312

6.5 Creating Pseudonyms

2313 Pseudonyms are created in a pseudonym service using the WS-Resource “CREATE” operation with the
2314 [WS-ResourceTransfer] extensions. This allows one or more pseudonyms to be added. The dialect
2315 defined in 6.1 (or the <fed:FilterPseudonyms> header) is specified on the CREATE to only create
2316 pseudonym information corresponding to the filter. If a filter is not specified, then the CREATE impacts
2317 the full pseudonym set. It is RECOMMENDED that filters be used.

2318 The following example creates pseudonym associated with the identity (indicated binary security token)
2319 for the locality (target scope) indicated by the URI <http://www.fabrikam123.com/NNK>.

2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331

```
<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/wsrfed/federation/200706/pseudonymdialect">  
        <wsrt:Fragment>  
          <wsrt:Expression>  
            <fed:FilterPseudonyms>  
              <fed:PseudonymBasis>  
                <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>  
              </fed:PseudonymBasis>  
              <fed:RelativeTo>
```

```
2332         <wsa:Address>
2333             http://www.fabrikam123.com/NNK
2334         </wsa:Address>
2335     </fed:RelativeTo>
2336 </fed:FilterPseudonyms>
2337 </wsrt:Expression>
2338 <wsrt:Value>
2339     <fed:Pseudonym>
2340         <fed:PseudonymBasis>
2341             <wsse:BinarySecurityToken>...</wsse:BinarySecurityToken>
2342         </fed:PseudonymBasis>
2343         <fed:RelativeTo>
2344             <wsa:Address>
2345                 http://www.fabrikam123.com/NNK
2346             </wsa:Address>
2347         </fed:RelativeTo>
2348         <fed:SecurityToken>
2349             <wsse:UsernameToken>
2350                 <wsse:Username> "Nick" </wsse:Username>
2351             </wsse:UsernameToken>
2352         </fed:SecurityToken>
2353         <fed:ProofToken>...</fed:ProofToken>
2354     </fed:Pseudonym>
2355 </wsrt:Value>
2356 </wsrt:Fragment>
2357 </wsrt:Create>
2358 </S:Body>
2359 </S:Envelope>
```

2360

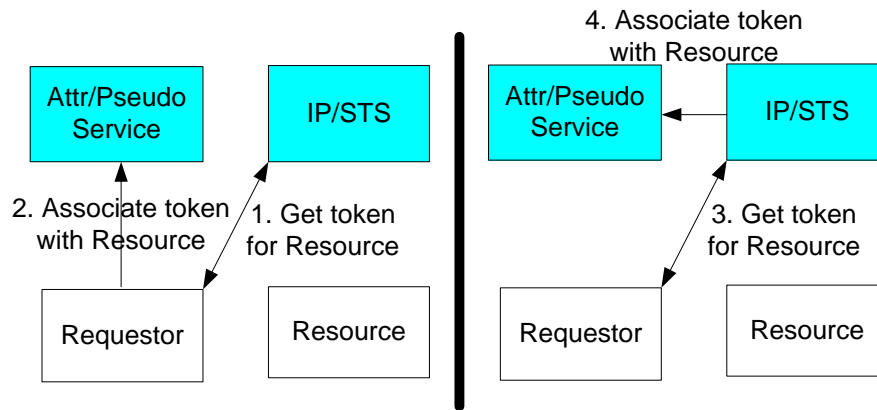
7 Security Tokens and Pseudonyms

2361
2362
2363
2364

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.

2365
2366
2367
2368
2369
2370
2371

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).



2372

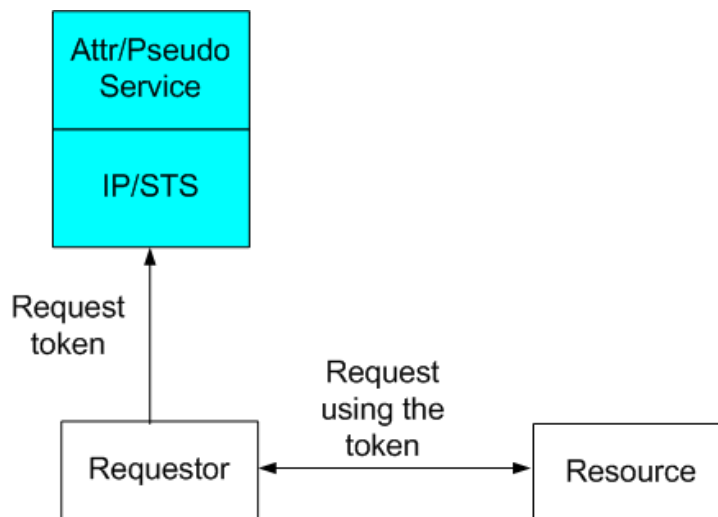
2373

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

2374

2375
2376
2377
2378

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:



2379

2380

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

2381 The pseudonym service SHOULD be self-maintained with respect to valid security tokens. That is,
2382 security tokens that have expired or are otherwise not valid for any reason MAY be automatically
2383 discarded by the service.

2384 This approach is an alternative to having the pseudonym service directly return the security token
2385 issuance. Both approaches SHOULD be supported in order to address different scenarios and
2386 requirements.

2387 The following sub-sections describe how token issuance works for different types of keys.

2388 7.1 RST and RSTR Extensions

2389 With the addition of pseudonyms and the integration of an IP/STS with a pseudonym service, an IP/STS
2390 MAY automatically map pseudonyms based on the target service. If it doesn't, the following additional
2391 options MAY be included in the security token requests using the `<wst:RequestSecurityToken>`
2392 request to explicitly request a mapping or to clarify the type of mapping desired.

2393 The following syntax illustrates the RST extension to support these new options:

```
2394 <fed:RequestPseudonym SingleUse="xs:boolean" ? Lookup="xs:boolean" ? ...>  
2395   ...  
2396 </fed:RequestPseudonym>
```

2397 `/fed:RequestPseudonym`

2398 This OPTIONAL element MAY be specified in a `<wst:RequestSecurityToken>` request to
2399 indicate how pseudonyms are to be processed for the requested token.

2400 `/fed:RequestPseudonym/@SingleUse`

2401 This optional OPTIONAL attribute indicates if a single-use pseudonym is returned (true), or if the
2402 service uses a constant identifier (false – the default).

2403 `/fed:RequestPseudonym/@Lookup`

2404 This OPTIONAL attribute indicates if an associated pseudonym for the specified scope is used
2405 (true – the default) or if the primary identity is used even if an appropriate pseudonym is
2406 associated (false).

2407 `/fed:RequestPseudonym/{any}`

2408 This is an extensibility mechanism to allow additional information to be specified. Use of this
2409 extensibility mechanism MUST NOT alter the semantics defined in this specification.

2410 `/fed:RequestPseudonym/@{any}`

2411 This is an extensibility mechanism to allow additional attributes to be specified. Use of this
2412 extensibility mechanism MUST NOT alter the semantics defined in this specification.

2413 If the `<RequestPseudonym>` isn't present, pseudonym usage/lookup and single use is at the discretion
2414 of the IP/STS. Note that if present, as with all RST parameters, processing is at the discretion of the STS
2415 and it MAY choose to use its own policy instead of honoring the requestor's parameters.

2416 Note that the above MAY be echoed in a RSTR response confirming the value used by the STS.

2417 7.2 Usernames and Passwords

2418 If an IP/STS returns a security token based on a username, then the token can be stored in the
2419 pseudonym service.

2420 If a corresponding password is issued (or if the requestor specified one), then it too MAY be stored with
2421 the pseudonym and security token so that it can be returned as the proof-of-possession token in the
2422 RSTR response.

2423 If a pseudonym is present, but no security token is specified, then the IP/STS MAY return a
2424 <UsernameToken> in the RSTR response to indicate the pseudonym.

2425 **7.3 Public Keys**

2426 Generally, when an IP/STS issues a new security token with public key credentials, the public key in the
2427 new security token is the same as the key in the provided input security token thereby allowing the same
2428 proof (private key) to be used with the new token since the public key is the same. In such cases, the
2429 new token can be saved directly.

2430 If, however, the IP/STS issues a new public key (and corresponding private key), then the private key
2431 MAY be stored with the pseudonym as a proof token so that it can be subsequently returned as the proof-
2432 of-possession token in the RSTR response.

2433 **7.4 Symmetric Keys**

2434 If an IP/STS returns a token based on a symmetric key (and the corresponding proof information), then
2435 the proof information MAY be stored with the pseudonym and token so that it can be used to construct a
2436 proof-of-possession token in the RSTR response.

2437

8 Additional WS-Trust Extensions

2438

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

2439

8.1 Reference Tokens

2440

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.

2441

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.

2442

2443

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.

2444

2445

2446

2447

2448

2449

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

2450

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

2451

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

2452

2453

2454

2455

2456

2457

2458

2459

```
<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>
```

2460

/fed:ReferenceToken

2461

2462

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.

2463

/fed:ReferenceToken/fed:ReferenceEPR

2464

2465

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

2466

/fed:ReferenceToken/fed:ReferenceDigest

2467

2468

2469

2470

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.

2471

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

2472

2473

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

2474

/fed:ReferenceToken/fed:ReferenceType

2475

2476

2477

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.

2478

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

2479 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2480 mechanism MUST NOT alter the semantics defined in this specification.

2481 /fed:ReferenceToken/fed:SerialNo

2482 An OPTIONAL URI value that uniquely identifies the reference token.

2483 /fed:ReferenceToken/fed:SerialNo/{any}

2484 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2485 mechanism MUST NOT alter the semantics defined in this specification.

2486 /fed:ReferenceToken/{any}

2487 This extensibility mechanism allows additional informative elements to be specified Use of this
2488 extensibility mechanism MUST NOT alter the semantics defined in this specification.

2489 /fed:ReferenceToken/{any}

2490 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2491 mechanism MUST NOT alter the semantics defined in this specification.

2492 There are no requirements on the security associated with the handle or dereferencing it. If the resulting
2493 token is secured or does not contain sensitive information the STS MAY just make it openly accessible.
2494 Alternatively, the STS MAY use the <wsp:AppliesTo> information from the RST to secure the token
2495 such that only requestors that can speak for that address can obtain the token.

2496 8.2 Indicating Federations

2497 In some scenarios an STS, resource provider, or service provider MAY be part of multiple federations and
2498 allow token requests at a single endpoint that could be processed in the context of any of the federations
2499 (so long as the requestor is authorized). In such cases, there may be a need for the requestor to identify
2500 the federation context in which it would like the token request to be processed.

2501 The following <fed:FederationID> element can be included in a RST (as well as an RSTR):

```
2502 <fed:FederationID ...>xs:anyURI</fed:FederationID>
```

2503 /fed:FederationID

2504 This element identifies the federation context as a URI value in which the token request is made
2505 (or was processed).

2506 /fed:FederationID/{any}

2507 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2508 mechanism MUST NOT alter the semantics defined in this specification.

2509 Note that if a FederationID is not specified, the *default* federation is assumed.

2510 8.3 Obtaining Proof Tokens from Validation

2511 A requestor may obtain a token for a federation for which the recipient service doesn't actually have the
2512 rights to use and extract the session key. For example, when a requestor's IP/STS and the recipient's
2513 IP/STS have an arrangement and share keys but the requestor and recipient only describe federation
2514 between themselves. In such cases, the requestor and the recipient MUST obtain the session keys
2515 (proof tokens) from their respective IP/STS. For the requestor this is returned in the proof token of its
2516 request.

2517 For the recipient, it must pass the message to its IP/STS to have it validated. As part of the validation
2518 process, the proof token MAY be requested by including the parameter below in the RST. When this
2519 element is received by an IP/STS, it indicates a desire to have a <wst:RequestedProofToken>
2520 returned with the session key so that the recipient does not have to submit subsequent messages for
2521 validation.

2522 The syntax of the `<fed:RequestProofToken>` is as follows:

```
2523 <fed:RequestProofToken ...>
2524   ...
2525 </fed:RequestProofToken>
```

2526 `/fed:RequestProofToken`

2527 When used with a *Validate* request this indicates that the requestor would like the STS to return a
2528 proof token so that subsequent messages using the same token/key can be processed by the
2529 recipient directly.

2530 `/fed:RequestProofToken/@{any}`

2531 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2532 mechanism MUST NOT alter the semantics defined in this specification.

2533 `/fed:RequestProofToken/{any}`

2534 This contents of this element are undefined and MAY be extended. Use of this extensibility
2535 mechanism MUST NOT alter the semantics defined in this specification.

2536

2537 8.4 Client-Based Pseudonyms

2538 Previous sections have discussed requesting pseudonyms based on registered identities. In some cases
2539 a requestor desires a pseudonym to be issued using *ad hoc* data that is specifies as an extension to the
2540 RST request. As with all WS-Trust parameters, the IP/STS is NOT REQUIRED to honor the parameter,
2541 but if it does, it SHOULD echo the parameter in the RSTR.

2542 A requestor MAY specify the `<fed:ClientPseudonym>` element to indicate pseudonym information it
2543 would like used in the issued token. The STS MUST accept all of the information or none of it. That is, it
2544 MUST NOT use some pseudonym information but not other pseudonym information.

2545 The syntax of the `<fed:ClientPseudonym>` element is as follows:

```
2546 <fed:ClientPseudonym ...>
2547   <fed:PPID ...>xs:string</fed:PPID> ?
2548   <fed:DisplayName ...>xs:string</fed:DisplayName> ?
2549   <fed:Email ...>xs:string</fed:EMail> ?
2550   ...
2551 </fed:ClientPseudonym>
```

2552 `/fed:ClientPseudonym`

2553 This indicates a request to use specific identity information in resulting security tokens.

2554 `/fed:ClientPseudonym/fed:PPID`

2555 If the resulting security token contains any form of private personal identifier, this string value is to
2556 be used as the basis. The issuer MAY use this value as the input (a seed) to a custom function
2557 and the result used in the issued token.

2558 `/fed:ClientPseudonym/fed:PPID/@{any}`

2559 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2560 mechanism MUST NOT alter the semantics defined in this specification.

2561 `/fed:ClientPseudonym/fed:DisplayName`

2562 If the resulting security token contains any form of display or subject name, this string value is to
2563 be used.

2564 `/fed:ClientPseudonym/fed:DisplayName/@{any}`

2565 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2566 mechanism MUST NOT alter the semantics defined in this specification.

2567 /fed:ClientPseudonym/fed:EMail

2568 If the resulting security token contains any form electronic mail address, this string value is to be
2569 used.

2570 /fed:ClientPseudonym/fed:EMail/{any}

2571 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2572 mechanism MUST NOT alter the semantics defined in this specification.

2573 /fed:ClientPseudonym/{any}

2574 This extensibility point allows other pseudonym information to be specified. If the STS does not
2575 understand any element it MUST either ignore the entire <fed:ClientPseudonym> or Fault.

2576 /fed:ClientPseudonym/{any}

2577 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2578 mechanism MUST NOT alter the semantics defined in this specification.

2579 8.5 Indicating Freshness Requirements

2580 There are times when a token requestor desires to limit the age of the credentials used to authenticate.
2581 The parameter MAY be specified in a RST to indicate the desired upper bound on credential age. As well
2582 this parameter is used to indicate if the requestor is willing to allow issuance based on cached
2583 credentials.

2584 The syntax of the <fed:Freshness> element is as follow:

```
2585 <fed:Freshness AllowCache="xs:boolean" ...>  
2586   xs:unsignedInt  
2587 </fed:Freshness>
```

2588 /fed:Freshness

2589 This indicates a desire to limit the age of authentication credentials. This REQUIRED unsigned
2590 integer value indicates the upper bound on credential age specified in minutes only. A value of
2591 zero (0) indicates that the STS is to immediately verify identity if possible or use the minimum age
2592 credentials possible if immediate (e.g. interactive) verification is not possible. If the `AllowCache`
2593 attribute is specified, then the cached credentials SHOULD meet the freshness time window.

2594 /fed:Freshness/{any}

2595 This extensibility mechanism allows additional attributes to be specified. Use of this extensibility
2596 mechanism MUST NOT alter the semantics defined in this specification.

2597 /fed:Freshness/@AllowCache

2598 This OPTIONAL Boolean qualifier indicates if cached credentials are allowed. The default value
2599 is *true* indicating that cached information MAY be used. If *false* the STS SHOULD NOT use
2600 cached credentials in processing the request.

2601 If the credentials provided are valid but do not meet the freshness requirements, then the
2602 `fed:NeedFresherCredentials` fault MUST be returned informing the requestor that they need to
2603 obtain fresher credentials in order to process their request.

2604 9 Authorization

2605 An authorization service is a specific instance of a security token service (STS). To ensure consistent
2606 processing and interoperability, this specification defines a common model for authorization services, a
2607 set of extensions enabling rich authorization, and a common profile of [WS-Trust] to facilitate
2608 interoperability with authorization services.

2609 This section describes a model and two extensions specific to rich authorization. The first allows
2610 additional context information to be provided in authorization requests. The second allows services to
2611 indicate that additional claims are required to successfully process specific requests.

2612 9.1 Authorization Model

2613 An authorization service is an STS that operates in a decision brokering process. That is, it receives a
2614 request (either directly or on behalf of another party) for a token (or set of tokens) to access another
2615 service. Such a service MAY be separate from the target service or it MAY be co-located. The
2616 authorization service determines if the requested party can access the indicated service and, if it can,
2617 issues a token (or set of tokens) with the allowed rights at the specified service. These two aspects are
2618 distinct and could be performed by different collaborating services.

2619 In order to make the authorization decision, the authorization service MUST ensure that the requestor has
2620 presented and proven the claims required to access the target service (or resource) indicated in the
2621 request (e.g. in the `<wsp:AppliesTo>` parameter). Logically, the authorization service constructs a
2622 table of name/value pairs representing the claims required by the target service. The logical *requirement*
2623 *table* is constructed from the following sources and may MAY be supplemented by additional service
2624 resources:

- 2625 • The address of the EPR for the target service
- 2626 • The reference properties from the EPR of the target service
- 2627 • Parameters of the RST
- 2628 • External access control policies

2629 Similarly, the claim table is a logical table representing the claims and information available for the
2630 requestor that the authorization service uses as the basis for its decisions. This logical table is
2631 constructed from the following sources:

- 2632 • Proven claims that are bound to the RST request (both primary and supporting)
- 2633 • Supplemental authorization context information provided in the request
- 2634 • External authorization policies

2635 9.2 Indicating Authorization Context

2636 In the [WS-Trust] protocol, the requestor of a token conveys the desired properties of the required token
2637 (such as the token type, key type, claims needed, etc.) in the token request represented by the RST
2638 element. Each such property is represented by a child element of the RST, and is typically specified by
2639 the Relying Party that will consume the issued token in its security policy assertion as defined by [WS-
2640 SecurityPolicy]. The token properties specified in a token request (RST) generally translate into some
2641 aspect of the content of the token that is issued by a STS. However, in many scenarios, there is a need to
2642 be able to convey additional contextual data in the token request that influences the processing and token
2643 issuance behavior at the STS. The supplied data MAY (but need not) directly translate into some aspect
2644 of the actual token content.

2645 To enable this a new element, `<auth:AdditionalContext>`, is defined to provide additional context
 2646 information. This MAY be specified in RST requests and MAY be included in RSTR responses.

2647 The syntax is as follows:

```

2648 <wst:RequestSecurityToken>
2649   ...
2650   <auth:AdditionalContext>
2651     <auth:ContextItem Name="xs:anyURI" Scope="xs:anyURI" ? ...>
2652       (<auth:Value>xs:string</auth:Value> |
2653        xs:any ) ?
2654     </auth:ContextItem> *
2655     ...
2656   </auth:AdditionalContext>
2657   ...
2658 </wst:RequestSecurityToken>
  
```

2659 The following describes the above syntax:

2660 `/auth:AdditionalContext`

2661 This OPTIONAL element provides additional context for the authorization decision (which
 2662 determines token issuance).

2663 `/auth:AdditionalContext/ContextItem`

2664 This element is provides additional authorization context as simple name/value pairs. Note that
 2665 this is the only `fed:AdditionalContext` element defined in this specification.

2666 `/auth:AdditionalContext/ContextItem/@Name`

2667 This REQUIRED URI attribute specifies the kind of the context item being provided. There are no
 2668 pre-defined context names.

2669 `/auth:AdditionalContext/ContextItem/@Scope`

2670 This OPTIONAL URI attribute specifies the scope of the context item. That is, the subject of the
 2671 context item. If this is not specified, then the scope is undefined.

2672 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

2673 `/auth:AdditionalContext/ContextItem/Value`

2674 This OPTIONAL string element specifies the simple string value of the context item.

2675 `/auth:AdditionalContext/ContextItem/{any}`

2676 This OPTIONAL element allows a custom context value to be associated with the context item.
 2677 This MUST NOT be specified along with the `Value` element (there can only be a single value).

2678 /auth:AdditionalContext/ContextItem/@{any}

2679 This extensibility point allows additional attributes to be specified. Use of this extensibility
2680 mechanism MUST NOT violate any semantics defined in this document.

2681 /auth:AdditionalContext/@{any}

2682 This extensibility point allows additional attributes. Use of this extensibility mechanism MUST
2683 NOT violate any semantics defined in this document.

2684 /auth:AdditionalContext/{any}

2685 This element has an open content model allowing different types of context to be specified. That
2686 is, custom elements can be defined and included so long as all involved parties understand the
2687 elements.

2688 An example of an RST token request where this element is used to specify additional context data is
2689 given below. Note that this example specifies claims using a custom dialect.

```
2690 <wst:RequestSecurityToken>
2691   <wst:TokenType>
2692     urn:oasis:names:tc:SAML:1.0:assertion
2693   </wst:TokenType>
2694   <wst:RequestType>
2695     http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
2696   </wst:RequestType>
2697   <wst:Claims Dialect="...">
2698     ...
2699   </wst:Claims>
2700   ...
2701   <auth:AdditionalContext>
2702     <auth:ContextItem Name="urn:...:PurchaseAmount">
2703       <auth:Value>125.00</auth:Value>
2704     </auth:ContextItem>
2705     <auth:ContextItem Name="urn:...:MerchantId">
2706       <auth:Value>FABRIKAM 92305645883256</auth:Value>
2707     </auth:ContextItem>
2708   </auth:AdditionalContext>
2709 </wst:RequestSecurityToken>
```

2710 9.3 Common Claim Dialect

2711 There are different claim representations that are used across different Web Service implementations
2712 making it difficult to express claims in a common interoperable way. To facilitate interoperability, this
2713 section defines a simple dialect for expressing claims in a format-neutral way. This new dialect uses the
2714 <auth:ClaimType> element for representing a claim, and the dialect is identified by the following URI:

```
2715 http://docs.oasis-open.org/wsfed/authorization/200706/authclaims
```

2716 This dialect MAY be used within the <wst:Claims> element when making token requests or in
2717 responses. This dialect MAY also be used in describing a service's security requirements using [WS-
2718 SecurityPolicy]. Note that the xml:lang attribute MAY be used where allowed via attribute extensibility to
2719 specify a language of localized elements and attributes using the language codes specified in [RFC
2720 3066].

2721 The syntax for the <auth:ClaimType> element for representing a claim is as follows:

```
2722 <auth:ClaimType Uri="xs:anyURI" Optional="xs:boolean">
2723   <auth:DisplayName ...> xs:string </auth:DisplayName> ?
2724   <auth:Description ...> xs:string </auth:Description> ?
2725   <auth:DisplayValue ...> xs:string </auth:DisplayValue> ?
2726   (<auth:Value>...</auth:Value> |
2727   <auth:StructuredValue ...>...</auth:StructuredValue> |
```

2728
2729
2730
2731
2732
2733

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

2734 The following describes the above syntax:

2735 /auth:ClaimType

2736 This element represents a specific claim.

2737 /auth:ClaimType/@Uri

2738 This REQUIRED URI attribute specifies the kind of the claim being indicated. The following claim
2739 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

2740 /auth:ClaimType/@Optional

2741 This OPTIONAL boolean attribute specifies the claim is optional (true) or required (false). The
2742 default value is false.

2743 /auth:ClaimType/auth:DisplayName

2744 This OPTIONAL element provides a friendly name for this claim type that can be shown in user
2745 interfaces.

2746 /auth:ClaimType/auth:DisplayName/@{any}

2747 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
2748 NOT alter the semantics defined in this specification.

2749 /auth:ClaimType/auth:Description

2750 This OPTIONAL element provides a description of the semantics for this claim type.

2751 /auth:ClaimType/auth:Description/@{any}

2752 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
2753 NOT alter the semantics defined in this specification.

2754 /auth:ClaimType/auth:DisplayValue

2755 This OPTIONAL element provides a displayable value for a claim returned in a security token.

2756 /auth:ClaimType/auth:DisplayValue/@{any}

2757 This extensibility point allows attributes to be added. Use of this extensibility mechanism MUST
2758 NOT alter the semantics defined in this specification.

2759 /auth:ClaimType/auth:Value

2760 This OPTIONAL element allows a specific string value to be specified for the claim.

2761 /auth:ClaimType/auth:EncryptedValue

2762 This OPTIONAL element is used to convey the ciphertext of a claim.

2763 /auth:Claims/auth:ClaimType/auth:EncryptedValue/xenc:EncryptedData

2764 This OPTIONAL element is only used for conveying the KeyInfo.

2765 /auth:Claims/auth:ClaimType/auth:EncryptedValue/@DecryptionCondition
 2766 This OPTIONAL attribute specifies the URI indicating the conditions under which this claim
 2767 SHOULD be decrypted.
 2768 The decryptor SHOULD decrypt only if the decryption condition is fulfilled. Note that a decryptor
 2769 MAY be a 3rd party. In order for such a decryption to happen, the recipient of the claim has to
 2770 provide the ciphertext and decryption condition to the decryptor.. This specification does not
 2771 define any URI values. Participating parties MAY use other values under private agreements.

2772 /auth:ClaimType/auth:StructuredValue
 2773 This OPTIONAL element specifies the value of a claim in a well formed xml structure.

2774 /auth:ClaimType/auth:StructuredValue/@{any}
 2775 This extensibility point allows additional structured value types to be specified for the claim. Use
 2776 of this extensibility point MUST NOT alter the semantics defined in this specification.

2777

2778 /auth:ClaimType/auth:ConstrainedValue
 2779 This OPTIONAL element specifies constraints on a given claim. It MAY contain the constraint that
 2780 value MUST satisfy, or it MAY contain the actual constrained value. For more details on
 2781 constraints see section 9.3.1.

2782 /auth:ClaimType/@{any}
 2783 This extensibility point allows attributes to be added. Use of this extensibility point MUST NOT
 2784 alter the semantics defined in this specification.

2785 /auth:ClaimType/{any}
 2786 This extensibility point allows additional values types to be specified for the claim. Use of this
 2787 extensibility point MUST NOT alter the semantics defined in this specification.

2788

2789 9.3.1 Expressing value constraints on claims

2790 When requesting or returning claims in a [WS-Trust] RST request or specifying required claims in [WS-
 2791 SecurityPolicy] it MAY be necessary to express specific constraints on those claims. The
 2792 <auth:ConstrainedValue> element, used within the <auth:ClaimType> element, provides this
 2793 capability.

2794

2795 The semantics of the comparison operators specified in the <auth:ConstrainedValue> element are
 2796 specific to the given claim type unless explicitly defined below.

2797

2798 The syntax for the <auth:ConstrainedValue> element, used within the <auth:ClaimType>
 2799 element, is as follows.

```

2800 <auth:ConstrainedValue AssertConstraint="xs:boolean">
2801   ( <auth:ValueLessThan>
2802     (<auth:Value> xs:string </auth:Value> |
2803     <auth:StructuredValue> xs:any </auth:StructuredValue>)
2804   </auth:ValueLessThan> |
2805   <auth:ValueLessThanOrEqual>
2806     (<auth:Value> xs:string </auth:Value> |
2807     <auth:StructuredValue> xs:any </auth:StructuredValue>)
2808   </auth:ValueLessThanOrEqual> |
2809   <auth:ValueGreaterThan>
2810     (<auth:Value> xs:string </auth:Value> |
2811     <auth:StructuredValue> xs:any </auth:StructuredValue>)
  
```

```

2812     </auth:ValueGreaterThan> |
2813     <auth:ValueGreaterThanOrEqual>
2814         (<auth:Value> xs:string </auth:Value> |
2815         <auth:StructuredValue> xs:any </auth:StructuredValue>)
2816     </auth:ValueGreaterThanOrEqual> |
2817     <auth:ValueInRange>
2818         <auth:ValueUpperBound>
2819             (<auth:Value> xs:string </auth:Value> |
2820             <auth:StructuredValue> xs:any </auth:StructuredValue>)
2821         </auth:ValueUpperBound>
2822         <auth:ValueLowerBound>
2823             (<auth:Value> xs:string </auth:Value> |
2824             <auth:StructuredValue> xs:any </auth:StructuredValue>)
2825         </auth:ValueLowerBound>
2826     </auth:ValueInRange> |
2827     <auth:ValueOneOf>
2828         (<auth:Value> xs:string </auth:Value> |
2829         <auth:StructuredValue> xs:any </auth:StructuredValue>) +
2830     </auth:ValueOneOf> ) ?
2831     ...
2832 </auth:ConstrainedValue> ?

```

2833 The following describe the above syntax

2834 /auth:ClaimType/auth:ConstrainedValue

2835 This OPTIONAL element indicates that there are constraints on the claim value. This element
2836 MUST contain one of the defined elements below when used in a RST/RSTR message. This
2837 element MAY be empty when used in the fed:ClaimTypesOffered element to describe a service's
2838 capabilities which means that any constrained value form, from he defined elements below, is
2839 supported for the claim type.

2840 /auth:ClaimType/auth:ConstrainedValue/@AssertConstraint

2841 This OPTIONAL attribute indicates that when a claim is issued the constraint itself is asserted
2842 (when true) or that a value that adheres to the condition is asserted (when false). The default
2843 value is true.

2844 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThan

2845 This OPTIONAL element indicates that the value of the claim MUST be less than the given value.

2846 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThan/auth:Value

2847 This element specifies the string value the claim MUST be less than.

2848 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThan/auth:StructuredValue

2849 This element specifies the value of a claim in a well formed xml structure the claim MUST be less
2850 than.

2851 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThanOrEqual

2852 This OPTIONAL element indicates that the value of the claim MUST be less than or equal to the
2853 given value.

2854 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThanOrEqua/auth:Value

2855 This element specifies the string value the claim MUST be less than or equal to.

2856 /auth:ClaimType/auth:ConstrainedValue/auth:ValueLessThanOrEqual/auth:StructuredValue

2857 This element specifies the value of a claim in a well formed xml structure the claim MUST be less
2858 than or equal to.

2859 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThan

2860 This OPTIONAL element indicates that the value of the claim MUST be greater than the given
2861 value.

2862 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThan/auth:Value
2863 This element specifies the string value the claim MUST be greater than.

2864 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThan/auth:StructuredValue
2865 This element specifies the value of a claim in a well formed xml structure the claim MUST be
2866 greater than.

2867 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThanOrEqual
2868 This OPTIONAL element indicates that the value of the claim MUST be greater than or equal to
2869 the given value.

2870 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThanOrEqual/auth:Value
2871 This element specifies the string value the claim MUST be greater than or equal to.

2872 /auth:ClaimType/auth:ConstrainedValue/auth:ValueGreaterThanOrEqual/auth:StructuredValue
2873 This element specifies the value of a claim in a well formed xml structure the claim MUST be
2874 greater than or equal to.

2875 /auth:ClaimType/auth:ConstrainedValue/auth:ValueInRange
2876 This OPTIONAL element indicates that the value of the claim MUST be in the specified range.
2877 The specified boundary values are included in the range.

2878 /auth:ClaimType/auth:ConstrainedValue/auth:ValueInRange/auth:ValueUpperBound
2879 This element specifies the upper limit on a given value.

2880 /auth:ClaimType/auth:ConstrainedValue/auth:ValueInRange/auth:ValueLowerBound
2881 This element specifies the lower limit on a given value.

2882 /auth:ClaimType/auth:ConstrainedValue/auth:ValueOneOf
2883 This element specifies a collection of values among which the value of claim MUST fall.

2884 /auth:ClaimType/auth:ConstrainedValue/auth:ValueOneOf/auth:Value
2885 This element specifies an allowed string value for the claim.

2886 /auth:ClaimType/auth:ConstrainedValue/auth:ValueOneOf/auth:StructuredValue
2887 This element specifies an allowed value for the claim in a well formed xml structure.

2888 /auth:ClaimType/auth:ConstrainedValue/{any}
2889 This extensibility point allows additional constrained value types to be specified for the claim..
2890 Use of this extensibility mechanism MUST NOT alter the semantics defined in this specification.

2891
2892

2893 9.4 Claims Target

2894 The @fed:ClaimsTarget attribute is defined for use on the wst:Claims element as a way to indicate the
2895 intended consumer of claim information .

2896 The syntax for @auth:ClaimsTarget is as follows.

```
2897 <wst:Claims fed:ClaimsTarget="..." ...>
2898   ...
2899 </wst:Claims>
```

2900 The following describes the above syntax.

2901

2902 /wst:Claims /@fed:ClaimsTarget

2903 This OPTIONAL attribute indicates the intended consumer of the claim information. If this
2904 attribute is not specified, then a default value is assumed. The predefined values are listed in the
2905 table below, but parties MAY use other values under private agreements. This attribute MAY be
2906 used if the context doesn't provide a default target or if a different target is required. This attribute
2907 MUST NOT appear in a RST or RSTR message defined in WS-Trust,

2908

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.

2909

2910

2911 9.5 Authorization Requirements

2912 Authorization requestors and issuing services (providers) compliant with this specification MUST conform
2913 to the rules described in this section when issuing RST requests and returning RSTR responses.

2914 *R001* – The authorization service MUST accept an `<wsp:AppliesTo>` target in the RST

2915 *R002* – The authorization service MUST specify an `<wsp:AppliesTo>` target in the RSTR if one is
2916 specified in the RST

2917 *R003* – The authorization service SHOULD encode the `<wsp:AppliesTo>` target in issued tokens if the
2918 token format supports it

- 2919 *R004* – The `<wsp:AppliesTo>` target for issued token MAY be for a broader scope than the scope
2920 specified in the RST but MUST NOT be narrower (as specified in WS-Trust)
- 2921 *R005* – The authorization service MUST accept reference properties in the `<wsp:AppliesTo>` target
- 2922 *R006* – The authorization service MUST accept the `<auth:AdditionalContext>` parameter
- 2923 *R007* – The authorization service MUST accept the claim dialect defined in this specification
- 2924 *R008* – The authorization service MAY ignore elements in the `auth:AdditionalContext` parameter if it
2925 doesn't recognize or understand them

2926

10 Indicating Specific Policy/Metadata

2927 When a requestor communicates with a recipient service there may be additional security requirements,
2928 beyond those in the general security policy or other metadata, that are required based on the specifics of
2929 the request. For example, if a request contains a “gold customer” custom message header to indicate
2930 customer classification (and routing), then proof that the requestor is a gold member may be required
2931 when the request is actually authorized. There may also be contextual requirements which are hard to
2932 express in a general policy. For example, if a requestor wants to submit a purchase, it may be required to
2933 present a token from a trusted source attesting that the requestor has the requisite funds.

2934 To address this scenario a mechanism is introduced whereby the recipient service MAY indicate to the
2935 requestor that additional security semantics apply to the request. The requestor MAY reconstruct the
2936 message in accordance with the new requirements if it can do so. In some cases the requestor may
2937 need to obtain additional tokens from an authorization or identity service and then reconstruct and
2938 resubmit the message.

2939 The mechanism defined by this specification that MAY be used to dynamically indicate that a specific
2940 policy or metadata applies to a specific request is to issue a specialized SOAP Fault. This fault indicates
2941 to the requestor that additional security metadata is REQUIRED. The new metadata, in its complete form
2942 (not a delta) is specified in the fault message using the WS-MetadataExchange format.

2943 The fault is the `fed:SpecificMetadata` and is specified as the fault code. The `<S:Detail>` of this
2944 fault contains a `mex:Metadata` element containing sections with the effective metadata for the endpoint
2945 processing this specific request.

2946 The following example illustrates a fault with embedded policy:

```
2947 <S:Envelope xmlns:S="..." xmlns:auth="..." xmlns:wst="..." xmlns:fed="..."
2948   xmlns:sp="..." xmlns:wsp="..." xmlns:mex="...">
2949   <S:Body>
2950     <S:Fault>
2951       <S:Code>
2952         <S:Value>fed:SpecificMetadata</S:Value>
2953       </S:Code>
2954       <S:Reason>
2955         <S:Text>Additional credentials required in order to
2956           perform operation. Please resubmit request with
2957           appropriate credentials.
2958         </S:Text>
2959       </S:Reason>
2960       <S:Detail>
2961         <mex:Metadata>
2962           <mex:MetadataSection
2963             Dialect='http://schemas.xmlsoap.org/ws/2004/09/policy'>
2964             <wsp:Policy>
2965               ...
2966             <sp:EndorsingSupportingTokens>
2967               <sp:IssuedToken>
2968                 <sp:Issuer>...</sp:Issuer>
2969                 <sp:RequestSecurityTokenTemplate>
2970                   <wst:Claims>
2971                     ...
2972                   </wst:Claims>
2973                   <auth:AdditionalContext>
2974                     ...
2975                   </auth:AdditionalContext>
2976                   ...

```

```
2977         </sp:RequestSecurityTokenTemplate>
2978     </sp:IssuedToken>
2979     </sp:EndorsingSupportingTokens>
2980     </wsp:Policy>
2981     </mex:MetadataSection>
2982 </mex:Metadata>
2983 </S:Detail>
2984 </S:Fault>
2985 </S:Body>
2986 </S:Envelope>
```

2987

11 Authentication Types

2988

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

2989

2990

2991

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

2992

2993

12 Privacy

2994 When a requestor contacts an authority to obtain a security token or to obtain authorization for an action it
2995 is often the case that information subject to personal or organizational privacy requirements MAY be
2996 presented in order to authorize the request. In such cases the authority MAY require the requestor to
2997 indicate the restrictions it expects on the use and distribution of sensitive information contained in tokens
2998 it obtains. In this document, this is referred to as a “disclosure constraint”. It should be noted that
2999 disclosure constraints may apply if the requestor is requesting tokens for itself or if the requestor is acting
3000 on behalf of another party.

3001 This specification describes how requestors can communicate their disclosure constraints to security
3002 token services using the [WS-Trust] protocol. It additionally facilitates the requestor’s compliance with
3003 such constraints by allowing it to request elevated data protection for some or all of the response and
3004 issued tokens. The disclosure constraint and protection elevation request are communicated using
3005 existing WS-Trust mechanisms as well as extensions defined in this specification.

3006 The WS-Trust specification describes how to request tokens as well as parameters to the token request
3007 (RST) for indicating how to encrypt proof information as well as algorithms to be used. The following sub-
3008 sections define extension parameters that MAY be specified in RST requests (and echoed in RSTR
3009 responses) to indicate additional privacy options which complement the existing WS-Trust parameters.

3010 12.1 Confidential Tokens

3011 The information contained within an issued token MAY be confidential or sensitive. Consequently, the
3012 requestor may wish to have this information protected (confidential) so that only the intended recipient of
3013 the resulting token (or tokens) can access the information.

3014 The [WS-Trust] specification describes how to indicate a key to use if any data in the token is to be
3015 encrypted, but doesn’t specify any mandates around when or what data is to be protected. This
3016 parameter indicates a protection requirement from the requestor (the STS MAY choose to protect data
3017 even if the requestor doesn’t mandate it).

3018 Any protected (encrypted) information is secured using the token specified in the `<wst:Encryption>`
3019 parameter or using a token the recipient knows to be correct for the request.

3020 The following parameters MAY be specified in an RST request (and echoed in an RSTR response) to
3021 indicate that potentially sensitive information in the token be protected:

```
3022 <wst:RequestSecurityToken>  
3023 ...  
3024 <priv:ProtectData ...>  
3025   <wst:Claims ...>...</wst:Claims> ?  
3026   ...  
3027 </priv:ProtectData>  
3028 ...  
3029 </wst:RequestSecurityToken>
```

3030 The following describes the above syntax:

3031 /priv:ProtectData

3032 This OPTIONAL parameter indicates that sensitive information in any resultant tokens MUST be
3033 protected (encrypted). If specific claims are identified they MUST be protected. The issuer MAY
3034 have an out-of-band agreement with the requestor as to what MUST be protected. If not, and if
3035 specific claims are not identified, the issuer MUST protect all claims. The issuer MAY choose to
3036 protect more than just the requested claims.

3037 /priv:ProtectData/@{any}

3038 This extensibility point allows additional attributes to be specified. Use of this extensibility
3039 mechanism MUST NOT violate any semantics defined in this document.

3040 /priv:ProtectData/wst:Claims

3041 This OPTIONAL element allows the requestor to indicate specific claims which, at a minimum,
3042 MUST be protected. This re-uses the claim specification mechanism from [WS-Trust]. Claims
3043 specified in this set MUST be protected. There is no requirement that all claims specified are in
3044 the issued token. That is, claims identified but not issued MAY be ignored by the STS.

3045 /priv:ProtectData/{any}

3046 This extensibility point allows additional content to be specified Use of this extensibility point
3047 MUST NOT violate any semantics defined in this document.

3048 12.2 Parameter Confirmation

3049 The RST request MAY contain a number of parameters indicating a requestor's disclosure constraints
3050 and data protection preferences. The STS MAY choose , (but is not required) to honor these
3051 preferences and MAY, (or might not) include selected parameters in any RSTR response.

3052 For privacy reasons a requestor may wish to (a) know if privacy preferences (or any RST parameter)
3053 were accepted or not, (b) what default parameter values were used, (c) require that privacy preferences
3054 (or any RST parameter) be honored, and (d) know what the STS is reporting in a token if it is protected
3055 and unreadable by the requestor.

3056 The following parameters MAY be specified in a RST request (and echoed in an RSTR response) to
3057 indicate to support these requirements:

```
3058 <wst:RequestSecurityToken>  
3059 ...  
3060 <priv:EnumerateParameters ...>  
3061 <xs:list itemType='xs:QName' />  
3062 </priv:EnumerateParameters>  
3063 <priv:FaultOnUnacceptedRstParameters ...>  
3064 ...  
3065 </priv:FaultOnUnacceptedRstParameters>  
3066 <priv:EnumerateAllClaims ...>  
3067 ...  
3068 <priv:EnumerateAllClaims ...>  
3069 ...  
3070 </wst:RequestSecurityToken>
```

3071 The following describes the above syntax:

3072 /priv:EnumerateParameters

3073 A RST request MAY include parameters but the STS is not required to honor them. As such
3074 there is no way for the requestor to know what values were used by the STS. This OPTIONAL
3075 parameter provides a way to request the STS to return the values it used for parameters (or Fault
3076 if it refuses) – either taken from the RST or defaulted using internal policy or settings. The
3077 contents of this parameter indicate a list of QNames that represents RST parameters which
3078 MUST be included in the RSTR. That is, each QName listed MUST be present in the RSTR
3079 returned by the STS indicating the value the STS used for the parameter.

3080 /priv:EnumerateParameters/@{any}

3081 This extensibility point allows additional attributes to be specified. Use of this extensibility point
3082 MUST NOT violate any semantics defined in this document.

3083 /priv:FaultOnUnacceptedRstParameters

3084 This OPTIONAL parameter indicates that if any parameters specified in the RST are not accepted
3085 by the STS, then the STS MUST Fault the request (see the Error Code section for the applicable
3086 Fault code). This means that any unknown parameter causes the request to fail. Note that this
3087 includes extension parameters to the RST.

3088 /priv:FaultOnUnacceptedRstParameters/{any}

3089 This extensibility point allows additional attributes to be specified. Use of this extensibility point
3090 MUST NOT violate any semantics defined in this document.

3091 /priv:FaultOnUnacceptedRstParameters/{any}

3092 This extensibility point allows additional content to be specified. Use of this extensibility point
3093 MUST NOT violate any semantics defined in this document.

3094 /priv:EnumerateAllClaims

3095 This OPTIONAL parameter indicates that all claims issued in resulting tokens MUST be identified
3096 in the RSTR so that the requestor can inspect them. The claims are returned in a
3097 <wst:Claims> element in the RSTR.

3098 /priv:EnumerateAllClaims/{any}

3099 This extensibility point allows additional attributes to be specified. Use of this extensibility point
3100 MUST NOT violate any semantics defined in this document.

3101 /priv:EnumerateAllClaims/{any}

3102 This extensibility point allows additional content to be specified. Use of this extensibility point
3103 MUST NOT violate any semantics defined in this document.

3104 12.3 Privacy Statements

3105 Some services offer privacy statements. This specification defines a mechanism by which privacy
3106 statements, in any form of representation, can be obtained using the mechanisms defined in [WS-
3107 Transfer/WS-ResourceTransfer].

3108 The following URI is defined which can be used as a metadata section dialect in [WS-Transfer/WS-
3109 ResourceTransfer]:

```
3110 http://docs.oasis-open.org/wsfed/privacy/200706/privacypolicy
```

3111 As well, the following element can be used to indicate the EPR to which a [WS-Transfer/WS-
3112 ResourceTransfer] GET message can be sent to obtain the privacy policy:

```
3113 <priv:PrivacyPolicyEndpoint SupportsMex="xs:boolean" ?>  
3114   ...endpoint reference value...  
3115 </priv:PrivacyPolicyEndpoint
```

3116 This element is an endpoint-reference as described in [WS-Addressing]. A [WS-Transfer/WS-
3117 ResourceTransfer] GET message can be sent to it to obtain the previously defined privacy policy dialect.
3118 If the SupportsMex attribute is true (the default is false), then a [WS-MetadataExchange] request can be
3119 directed at the endpoint.

3120 Note that no specific privacy policy form is mandated so requestors must inspect the contents of the
3121 returned privacy policy (or policies) to determine if they can process it (them). The privacy policy could be
3122 a complete privacy policy document, a privacy policy document that references other privacy policies, or
3123 even a compact form of a privacy policy. The form of these documents is outside the scope of this
3124 document.

3125 Alternatively, HTTP GET targets can be specified by including a URL with the following federated
3126 metadata statement:

3127

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

3128

3129

13 Web (Passive) Requestors

3130 This specification defines a model and set of messages for brokering trust and federation of identity and
3131 authentication information across different trust realms and protocols. This section describes how this
3132 Federations model is applied to Web requestors such as Web browsers that cannot directly make Web
3133 Service requests.

3134 13.1 Approach

3135 The federation model previously described builds on the foundation established by [WS-Security] and
3136 [WS-Trust]. Typical Web client requestors cannot perform the message security and token request
3137 operations defined in these specifications. Consequently, this section describes the mechanisms for
3138 requesting, exchanging, and issuing security tokens within the context of a Web requestor.

3139 Web requestors use different but philosophically compatible message exchanges. For example, the
3140 resource might act as its own Security Token Service (STS) and not use a separate service (or even URI)
3141 thereby eliminating some steps. It is expected that subsequent profiles can be defined to extend the Web
3142 mechanisms to include additional exchange patterns.

3143 13.1.1 Sign-On

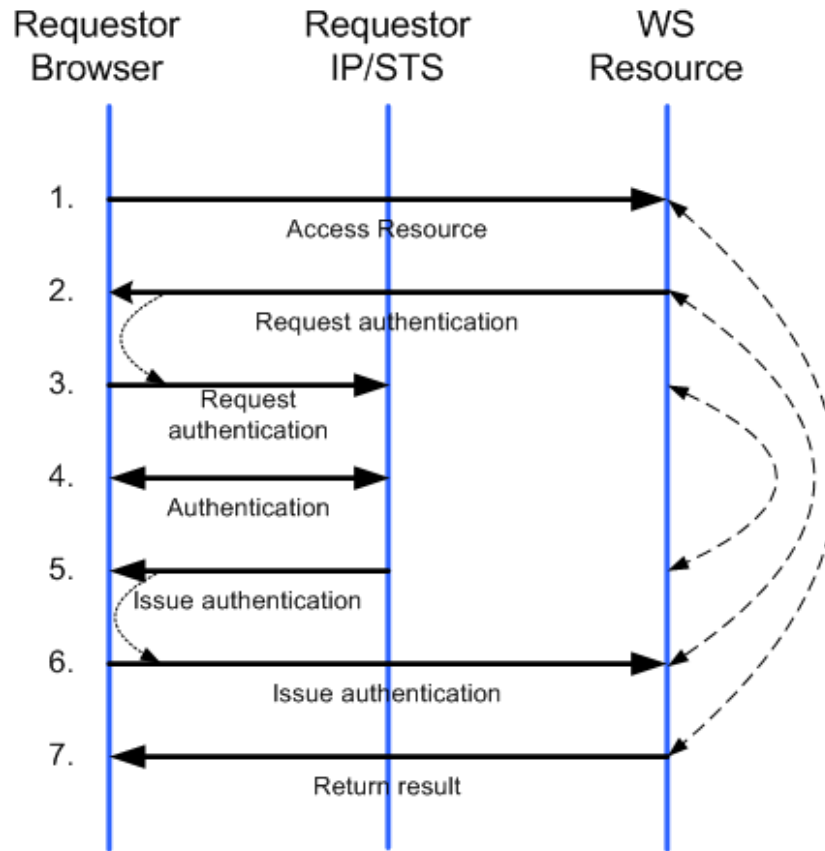
3144 The primary issue for *Web browsers* is that there is no easy way to directly issue SOAP requests.
3145 Consequently, the processing **MUST** be performed within the confines of the base HTTP 1.1 functionality
3146 (GET, POST, redirects, and cookies) and conform as closely as possible to the WS-Trust protocols for
3147 token acquisition.

3148 At a high-level, requestors are associated with an Identity Provider (IP) or Security Token Service (STS)
3149 where they authenticate themselves. At the time/point of initial authentication an artifact/cookie **MAY** be
3150 created for the requestor at their Identity Provider so that every request for a resource doesn't require
3151 requestor intervention. At other times, authentication at each request is the desired behavior.

3152 In the Web approach, there is a common pattern used when communicating with an IP/STS. In the first
3153 step, the requestor accesses the resource; the requestor is then redirected to an IP/STS if no token or
3154 cookie is supplied on the request. The requestor **MAY** be redirected to a local IP/STS operated by
3155 the resource provider. If it has not cached data indicating that the requestor has already been
3156 authenticated, a second redirection to the requestor's IP/STS will be performed. This redirection process
3157 **MAY** require prompting the user to determine the requestor's home realm. The IP/STS in the requestor's
3158 home realm generates a security token for use by the federated party. This token **MAY** be consumed
3159 directly by the resource, or it **MAY** be exchanged at the resource's IP/STS for a token consumable by the
3160 resource. In some cases the requestor's IP/STS has the requisite information cached to be able to issue
3161 a token, in other cases it must prompt the user. Note that the resource's IP/STS can be omitted if the
3162 resource is willing to consume the requestor's token directly.

3163 The figure below illustrates an example flow where there is no resource IP/STS. As depicted, all
3164 communication occurs with the standard HTTP GET and POST methods, using redirects (steps 2→3 and
3165 5→6) to automate the communication. Note that when returning non-URL content a POST is **REQUIRED**
3166 (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
3167 communication with an additional service isn't required. Note that step 3 depicts the resource redirecting
3168 directly to the requestor's IP/STS. As previously discussed, this could redirect to an IP/STS for the
3169 resource (or any number of chained IP/STS services). It might also redirect to a home realm discovery
3170 service.

3171 It should be noted that in step 4, the authentication protocol employed MAY be implementation-
3172 dependent.



3173

3174

Figure 25: Sample Browser Sign-On

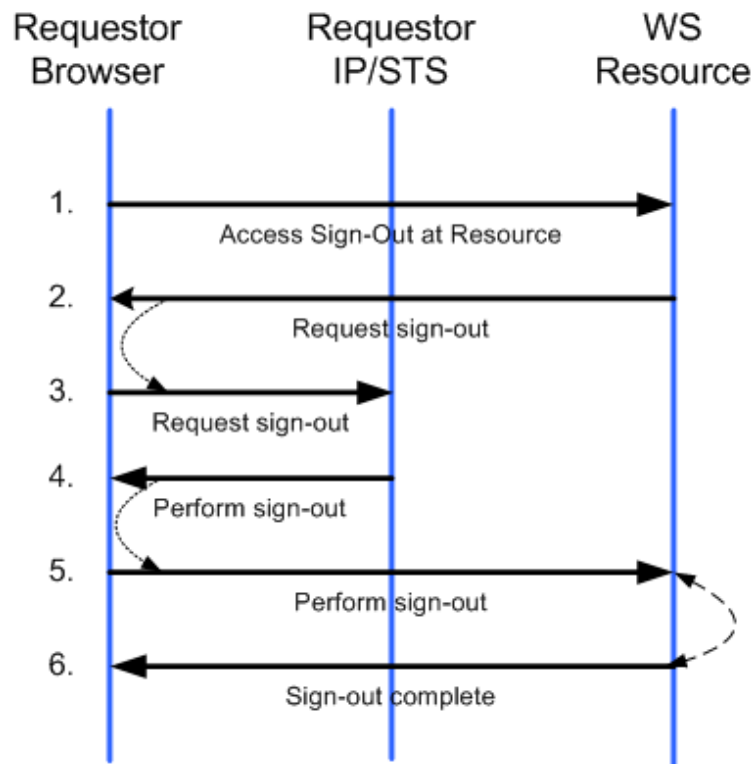
3175 13.1.2 Sign-Out

3176 For Web browsers, sign-out can be initiated by selecting the sign-out URL at a resource. In doing so, the
3177 browser will ultimately be redirected to the requestor's IP/STS indicating sign-out. Note that the browser
3178 MAY be first redirected to the resource's IP/STS and then to the requestor's IP/STS. Note that if multiple
3179 IP/STS services are used, and unaware of each other, multiple sign-outs MAY be required.

3180 The requestor's IP/STS SHOULD keep track of the realms to which it has issued tokens where cleanup
3181 may be required – specifically the IP/STS for the realms (or resources if different). When the sign-out is
3182 received at the requestor's IP/STS, it SHOULD initiate clean-up (e.g. issuing HTTP GET requests against
3183 the tracked realms indicating a sign-out cleanup is in effect or it can use the sign-out mechanism
3184 previously discussed). The exact mechanism by which this occurs is up to the IP/STS and is policy-
3185 driven. The only requirement is that a sign-out cleanup be performed at the IP/STS so that subsequent
3186 requests to the IP/STS don't use cached data.

3187 As described in section 4.2, there are two possible flows for these messages. They could be effectively
3188 chained through all the STSs involved in the session by successively redirecting the browser between
3189 each resource IP/STS and the requestor's IP/STS. Or the requestor's IP/STS can send sign-out
3190 messages to all the other STSs in parallel. The chained (sequential) approach has been found to be
3191 fragile in practice. If a resource IP/STS fails to redirect the user after cleaning up local state, or the
3192 network partitions, the sign-out notification will not reach all the resource IP/STSs involved. For this
3193 reason, compliant implementations SHOULD employ the parallel approach.

3194 When a sign-out clean-up GET is received at a realm, the realm SHOULD clean-up any cached
 3195 information and delete any associated artifacts/cookies. If requested, on completion the requestor is
 3196 redirected back to requestor's IP/STS.



3197

3198

Figure 26: Sample Browser Sign-Out

3199 The figure above illustrates this process where a resource-specific IP/STS doesn't exist. The mechanism
 3200 illustrated use redirection in steps 2 and 4 (optional) and the general *correlation* of messages to chain the
 3201 sign-out. As previously noted there could be a resource-specific IP/STS which handles local chaining or
 3202 notification.

3203 It should be noted that as a result of the single sign-out request (steps 5 and 6), an IP/STS MAY send
 3204 sign-out messages as described in this specification.

3205 13.1.3 Attributes

3206 At a high-level, attribute processing uses the same mechanisms defined for security token service
 3207 requests and responses. That is, redirection is used to issue requests to attribute services and
 3208 subsequent redirection returns the results of the attribute operations. All communication occurs with the
 3209 standard HTTP 1.1 GET and POST methods using redirects to automate the communication as shown in
 3210 the example below.

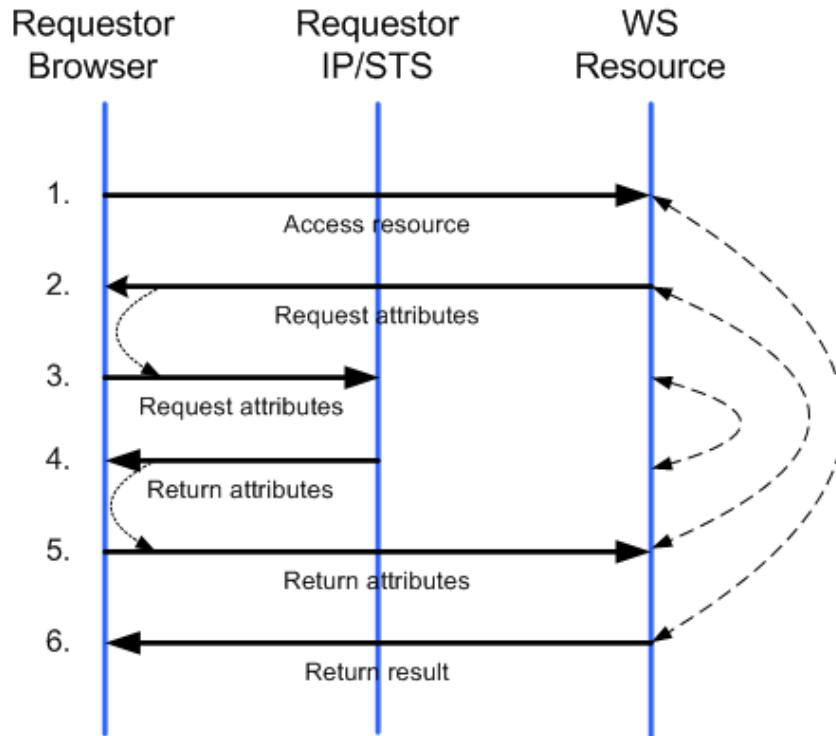


Figure 27: Sample Browser Attribute Access

3211

3212

3213 The figure above illustrates this process including calling out the redirection in steps 2 and 4 and the
 3214 general *correlation* of messages for an attribute scenario where there is no resource-specific IP/STS.

3215 As well, it should be noted that as a result of step 3 the IP/STS MAY prompt the user for approval before
 3216 proceeding to step 4.

3217 13.1.4 Pseudonyms

3218 At a high-level, pseudonym processing uses the same mechanisms defined for attribute and security
 3219 token service requests. That is, redirection is used to issue requests to pseudonym services and
 3220 subsequent redirection returns the results of the pseudonym operations. All communication occurs with
 3221 the standard HTTP GET and POST methods using redirects to automate the communication as in the
 3222 example below.

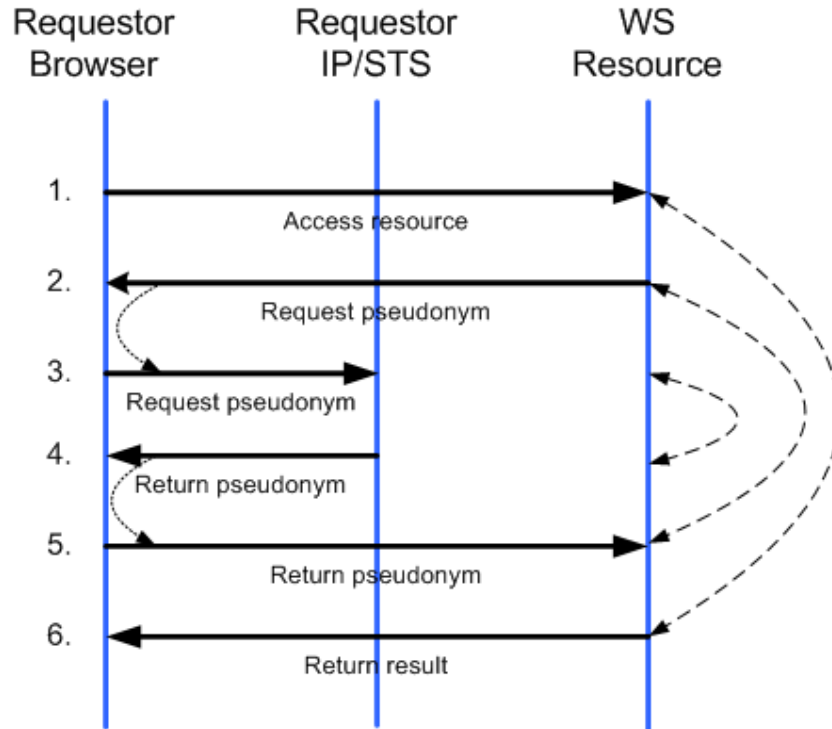


Figure 28: Sample Browser Pseudonym Access

3223

3224

3225 The figure above illustrates this process including calling out the redirection in steps 2 and 4 and the
 3226 general *correlation* of messages for an attribute scenario where there is no resource-specific IP/STS.

3227 13.1.5 Artifacts/Cookies

3228 In order to prevent requestor interaction on every request for security token, artifacts/cookies can be used
 3229 by SSO implementations as they are used today to cache state and/or authentication information or
 3230 issued tokens. However implementations MAY omit this caching if the desired behavior is to authenticate
 3231 on every request. As noted in the Security Consideration section later in this document, there are
 3232 security issues when using cookies.

3233 There are no restrictions placed on artifacts/cookie formats – they are up to each service to determine.
 3234 However, it is RECOMMENDED artifacts/cookies be encrypted or computationally hard to compromise.

3235 13.1.6 Bearer Tokens and Token References

3236 In cases where bearer tokens or references to tokens are passed it is strongly RECOMMENDED that the
 3237 messages use transport security in order to prevent attack.

3238 13.1.7 Freshness

3239 In cases where a resource requires specific authentication freshness, they can specify requirements in
 3240 their IP/STS requests, as described in the following section (see 13.2.2).

3241 13.2 HTTP Protocol Syntax

3242 This section describes the syntax of the protocols used by Web requestors. This protocol typically uses
3243 the redirection facilities of HTTP 1.1. This happens using a standard HTTP 302 error code for redirects
3244 (as illustrated below) and HTTP POST to push the forms:

```
3245 HTTP/1.1 302 Found  
3246 Location: url?parameters
```

3247 The exact parameters and form fields are described in detail in the sub-sections that follow the detailed
3248 example.

3249 In the descriptions below, some mechanisms are OPTIONAL meaning they MAY be supported. Within a
3250 mechanism, certain parameters MUST be specified while others, noted using square brackets, are
3251 OPTIONAL and MAY (or might not) be present.

3252 13.2.1 Parameters

3253 All HTTP 1.1 methods (both GET and POST) used in the redirection protocol allow query string
3254 parameters as illustrated below:

```
3255 GET url?parameters  
3256 POST url?parameters
```

3257 The GET and POST requests have required parameters and may have optional parameters depending
3258 on the operation being performed. For GET requests, these parameters are specified in the query string;
3259 for POST requests, these parameters are specified in the POST body (using the standard encoding rules
3260 for POST). The query string parameters of a POST request SHOULD be for extensibility only, and MAY
3261 be ignored by an implementation that is otherwise compliant with this specification.

3262 The following describes the parameters used for messages in this profile:

```
3263 wa=string  
3264 [wreply=URL]  
3265 [wres=URL]  
3266 [wctx=string]  
3267 [wp=URI]  
3268 [wct=timestring]  
3269 [wfed=string]  
3270 [wencoding=string]
```

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-in, this string MUST be "wsignin1.0".
3274 Note that this serves roughly the same purpose as the WS-Addressing Action header for the WS-
3275 Trust SOAP RST messages.

3276 wreply

3277 This OPTIONAL parameter is the URL to which responses are directed. Note that this serves
3278 roughly the same purpose as the WS-Addressing <wsa:ReplyTo> header for the WS-Trust
3279 SOAP RST messages.

3280 wres

3281 This OPTIONAL parameter is the URL for the resource accessed. This is a legacy parameter
3282 which isn't typically used. The *wrealm* parameter is typically used instead.

3283 wctx

3284 This OPTIONAL parameter is an opaque context value that MUST be returned with the issued
3285 token if it is passed in the request. Note that this serves roughly the same purpose as the WS-

3286 Trust SOAP RST @Context attribute. In order not to exceed URI length limitations, the value of
 3287 this parameter should be as small as possible.

3288 wp

3289 This OPTIONAL parameter is the URL for the policy which can be obtained using an HTTP GET
 3290 and identifies the policy to be used related to the action specified in "wa", but MAY have a
 3291 broader scope than just the "wa". Refer to WS-Policy and WS-Trust for details on policy and
 3292 trust. This attribute is only used to reference policy documents. Note that this serves roughly the
 3293 same purpose as the Policy element in the WS-Trust SOAP RST messages.

3294 wct

3295 This OPTIONAL parameter indicates the current time at the sender for ensuring freshness. This
 3296 parameter is the string encoding of time using the XML Schema datetime time using UTC
 3297 notation. Note that this serves roughly the same purpose as the WS-Security Timestamp
 3298 elements in the Security headers of the SOAP RST messages.

3299 wfed

3300 This OPTIONAL parameter indicates the federation context in which the request is made. This is
 3301 equivalent to the `FederationId` parameter in the RST message.

3302 wencoding

3303 This OPTIONAL parameter indicates the encoding style to be used for XML parameter content. If
 3304 not specified the default behavior is to use standard URL encoding rules. This specification only
 3305 defines one other alternative, `base64url` as defined in section 5 of [RFC 4648]. Support for
 3306 alternate encodings is expressed by assertions under the WebBinding assertion defined in this
 3307 specification.

3308 Note that any values specified in parameters are subject to encoding as specified in the HTTP 1.1
 3309 specification.

3310 When an HTTP POST is used, any of the query strings can be specified in the form contents using the
 3311 same name. Note that in this profile form values take precedence over URL parameters.

3312 Parameterization is extensible so that cooperating parties can exchange additional information in
 3313 parameters based on agreements or policy.

3314 13.2.2 Requesting Security Tokens

3315 The HTTP requests to an Identity Provider or security token service use a common syntax based on
 3316 HTTP forms. Requests typically arrive using the HTTP GET method as illustrated below but MAY be
 3317 issued using a POST method:

```
3318 GET resourceSTS?parameters HTTP/1.1
3319 POST resourceSTS?parameters HTTP/1.1
```

3320 The parameters described in the previous section (wa, wreply, wres, wctx, wp, wct) apply to the token
 3321 request. The additional parameters described below also apply. Note that any values specified in forms
 3322 are subject to encoding as described in the HTTP 1.1 specification.

3323 The following describes the additional parameters used for a token request:

```
3324 wtrealm=string
3325 [wfresh=freshness]
3326 [wauth=uri]
3327 [wreq=xml]
```

3328 wtrealm

3329 This REQUIRED parameter is the URI of the requesting realm. The wtrealm SHOULD be the
 3330 security realm of the resource in which nobody (except the resource or authorized delegates) can

3331 control URLs. Note that this serves roughly the same purpose as the AppliesTo element in the
3332 WS-Trust SOAP RST messages.

3333 wfresh

3334 This OPTIONAL parameter indicates the freshness requirements. If specified, this indicates the
3335 desired maximum age of authentication specified in minutes. An IP/STS SHOULD NOT issue a
3336 token with a longer lifetime. If specified as "0" it indicates a request for the IP/STS to re-prompt
3337 the user for authentication before issuing the token. Note that this serves roughly the same
3338 purpose as the Freshness element in the WS-Trust SOAP RST messages.

3339 wauth

3340 This OPTIONAL parameter indicates the REQUIRED authentication level. Note that this
3341 parameter uses the same URIs and is equivalent to the `wst:AuthenticationType` element in
3342 the WS-Trust SOAP RST messages.

3343 wreq

3344 This OPTIONAL parameter specifies a token request using either a
3345 `<wst:RequestSecurityToken>` element or a full request message as described in WS-Trust.
3346 If this parameter is not specified, it is assumed that the responding service *knows* the correct type
3347 of token to return. Note that this can contain the same RST payload as used in WS-Trust RST
3348 messages.

3349 To complete the protocol for requesting a token, it is necessary to redirect the Web requestor from the
3350 resource, or its local IP/STS, to the requestor's IP/STS. Determining the location of this IP/STS is
3351 frequently referred to as Home Realm Discovery; that is, determining the realm which manages the
3352 requestor's identity and thus where its IP/STS is located. This frequently involves interaction with the
3353 user (see section 13.5 for additional discussion). There are situations – particularly when users only
3354 access resources via portals and never directly via bookmarked URLs – when it can be advantageous to
3355 include the requestor's home realm in the request to avoid the requirement for human interaction. The
3356 following parameter MAY be specified for this purpose.

3357 `[whr=string]`

3358 whr

3359 This OPTIONAL parameter indicates the account partner realm of the client. This parameter is
3360 used to indicate the IP/STS address for the requestor. This may be specified directly as a URL or
3361 indirectly as an identifier (e.g. urn: or uuid:). In the case of an identifier the recipient is expected
3362 to know how to translate this (or get it translated) to a URL. When the *whr* parameter is used, the
3363 resource, or its local IP/STS, typically removes the parameter and writes a cookie to the client
3364 browser to remember this setting for future requests. Then, the request proceeds in the same
3365 way as if it had not been provided. Note that this serves roughly the same purpose as federation
3366 metadata for discovering IP/STS locations previously discussed.

3367 In the event that the XML request cannot be passed in the form (due to size or other considerations), the
3368 following parameter MAY be specified and the form made available by reference:

3369 `wreqptr=url`

3370 wreqptr

3371 This OPTIONAL parameter specifies a URL for where to find the request expressed as a
3372 `<wst:RequestSecurityToken>` element. Note that this does not have a WS-Trust parallel.
3373 The *wreqptr* parameter MUST NOT be included in a token request if *wreq* is present.

3374 When using *wreqptr* it is strongly RECOMMENDED that the provider of the *wreqptr* data authenticate the
3375 data to the consumer (relying party) in some way and that the provider authenticate consumers

3376 requesting the wreqptr data. If the wreqptr data is sensitive the provider SHOULD consider ensuring
3377 confidentiality of the data transfer.
3378 The RST is logically constructed to process the request. If one is specified (either directly via wreq or
3379 indirectly via wreqptr) it is the authoritative source for parameter information. That is, parameters outside
3380 of the RST (e.g. wfresh, wrealm, ...) are used to construct an RST if the RST is not present or if the
3381 corresponding RST values are not present.

3382 13.2.3 Returning Security Tokens

3383 Security tokens are returned by passing an HTTP form. To return the tokens, this profile embeds a
3384 <wst:RequestSecurityTokenResponse> element as specified in [WS-Trust].

```
3385 POST resourceURI?parameters HTTP/1.1  
3386 GET resourceURI?parameters HTTP/1.1
```

3387 In many cases the IP/STS to whom the request is being made, will prompt the requestor for information or
3388 for confirmation of the receipt of the token. As a result, the IP/STS can return an HTTP form to the
3389 requestor who then submits the form using an HTTP POST method. This allows the IP/STS to return
3390 security token request responses in the body rather than embedded in the limited URL query string.
3391 However, in some circumstances interaction with the requestor may not be required (e.g. cached
3392 information). In these circumstances the IP/STS have several options:

- 3393 1. Use a form anyway to confirm the action
- 3394 2. Return a form with script to automate and instructions for the requestor in the event that scripting
3395 has been disabled
- 3396 3. Use HTTP GET and return a pointer to the token request response (unless it is small enough to fit
3397 inside the query string)

3398 This specification RECOMMENDS using the POST method as the GET method requires additional state
3399 to be maintained and complicates the cleanup process whereas the POST method carries the state inside
3400 the method.

3401 Note that when using the POST method, any values specified in parameters are subject to encoding as
3402 described in the HTTP 1.1 specification. The standard parameters apply to returning tokens as do the
3403 following additional form parameters:

```
3404 wresult+xml  
3405 [wctx=string]
```

3406 wresult

3407 This REQUIRED parameter specifies the result of the token issuance. This can take the form of
3408 the <wst:RequestSecurityTokenResponse> element or
3409 <wst:RequestSecurityTokenResponseCollection> element, a SOAP security token
3410 request response (that is, a <S:Envelope>) as detailed in WS-Trust, or a SOAP <S:Fault>
3411 element. This carries the same content as a WS-Trust RSTR element (or even the actual SOAP
3412 Envelope containing the RSTR element).

3413 wctx

3414 This OPTIONAL parameter specifies the context information (if any) passed in with the request
3415 and typically represents context from the original request.

3416 In the event that the token/result cannot be passed in the form, the following parameter MAY be specified:

```
3417 wresultptr=url
```

3418 wresultptr

3419 This parameter specifies a URL to which an HTTP GET can be issued. The result is a document
3420 of type `text/xml` that contains the issuance result. This can either be the
3421 `<wst:RequestSecurityTokenResponse>` element, the
3422 `<wst:RequestSecurityTokenResponseCollection>` element, a SOAP response, or a
3423 SOAP `<S:Fault>` element. Note that this serves roughly the same purpose as the WS-
3424 ReferenceToken mechanism previously discussed (although this is used for the full response not
3425 just the token).

3426 13.2.4 Sign-Out Request Syntax

3427 This section describes how sign-out requests are formed and redirected by Web requestors. For
3428 modularity, it should be noted that support for sign-out is OPTIONAL.

3429 Sign-out can be initiated by a client at one of four points in the system:

- 3430 1. A Relying Party application server
- 3431 2. A Relying Party STS
- 3432 3. An application server local to the Identity Provider
- 3433 4. The Identity Provider STS

3434 For the first three use cases, the requestor's client must be redirected to the Identity Provider STS where
3435 the current session originated. This STS is required to send clean-up messages to all Relying Party STSs
3436 and any local applications for which the IP STS has issued security tokens for the requestor's current
3437 session. How the STS tracks this state for the requestor is implementation specific and outside the scope
3438 of this specification.

3439 As can be seen, for passive requestors the sign-out process is divided into two separate phases, referred
3440 to as sign-out and clean-up. Two different messages are used to ensure that all components of the
3441 system understand which phase is in effect to ensure that the requestor's sign-out request is processed
3442 correctly.

3443 13.2.4.1 Sign-out Message Syntax

3444

3445 The following describes the parameters used for the sign-out request (note that this parallels the sign-out
3446 SOAP message previously discussed):

```
3447 wa=string  
3448 wreply=URL
```

3449 wa

3450 This REQUIRED parameter specifies the action to be performed. By including the action, URIs
3451 can be overloaded to perform multiple functions. For sign-out, this string MUST be "wsignout1.0".

3452

3453 wreply

3454 This OPTIONAL parameter specifies the URL to return to once clean-up (sign-out) is complete. If
3455 this parameter is not specified, then after cleanup the GET completes by returning any realm-
3456 specific data such as a string indicating cleanup is complete for the realm.

3457 13.2.4.2 Clean-up Message Syntax

3458 The following describes the parameters used for the clean-up phase of a sign-out
3459 request:

3460
3461

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

3462 wa

3463 This required parameter specifies the action to be performed. By including the action, URIs can
3464 be overloaded to perform multiple functions. For the clean-up phase of a sign-out request, this
3465 string MUST be "wsignoutcleanup1.0".

3466 wreply

3467 This optional parameter specifies the URL to return to once clean-up is complete. If this
3468 parameter is not specified, then after cleanup the GET MAY complete by returning any realm-
3469 specific data such as a string indicating cleanup is complete for the realm.

3470

3471 13.2.5 Attribute Request Syntax

3472 This section describes how attribute requests are formed and redirected by Web requestors. For
3473 modularity, it should be noted that support for attributes is OPTIONAL. Additionally it should be noted
3474 that security considerations may apply. While the structure described here MAY be used with an attribute
3475 service supporting Web clients, the actual attribute request and response XML syntax is undefined and
3476 specific to the attribute store.

3477 The following describes the valid parameters used within attributes requests:

3478
3479
3480
3481
3482
3483
3484

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

3485 wa

3486 This REQUIRED parameter specifies the action to be performed. By including the action, URIs
3487 can be overloaded to perform multiple functions. For attribute requests, this string MUST be
3488 "wattr1.0".

3489 wreply

3490 This OPTIONAL parameter specifies the URL to return to when the attribute result is complete.

3491 wattr

3492 This OPTIONAL parameter specifies the attribute request. The syntax is specific to the attribute
3493 store being used and is not mandated by this specification. This attribute is only present on the
3494 request.

3495 wattrptr

3496 This OPTIONAL parameter specifies URL where the request can be obtained.

3497 wresult

3498 This OPTIONAL parameter specifies the result as defined by the attribute store and is not
3499 mandated by this specification. This attribute is only present on the responses.

3500 wresultptr

3501 This OPTIONAL parameter specifies URL where the result can be obtained.

3502 13.2.6 Pseudonym Request Syntax

3503 This section describes how pseudonym requests are formed and redirected by Web requestors. For
3504 modularity, it should be noted that support for pseudonyms is also OPTIONAL. As well, it should be
3505 noted that security considerations may apply.

3506 The following describes the valid parameters used within pseudonym requests (note that this parallels the
3507 pseudonym messages previously discussed):

```
3508 wa=string  
3509 [wreply=URL]  
3510 [wrealm=URL]  
3511 wpseudo=xml-pseudonym-request  
3512 wpseudoptr=URL  
3513 wresult=xml-result  
3514 wresultptr=URL
```

3515 wa

3516 This REQUIRED parameter specifies the action to be performed. By including the action, URIs
3517 can be overloaded to perform multiple functions. For pseudonym requests, this string MUST be
3518 "wpseudo1.0".

3519 wreply

3520 This OPTIONAL parameter specifies the URL to return to when the pseudonym result is
3521 complete.

3522 wpseudo

3523 This OPTIONAL parameter specifies the pseudonym request and either contains a SOAP
3524 envelope or a pseudonym request, such as a WS-Transfer/WS-ResourceTransfer <Get>. This
3525 attribute is only present on the request.

3526 wpseudoptr

3527 This OPTIONAL parameter specifies URL from which the request element can be obtained.

3528 wresult

3529 This OPTIONAL parameter specifies the result as either a SOAP envelope or a pseudonym
3530 response. This attribute is only present on the responses.

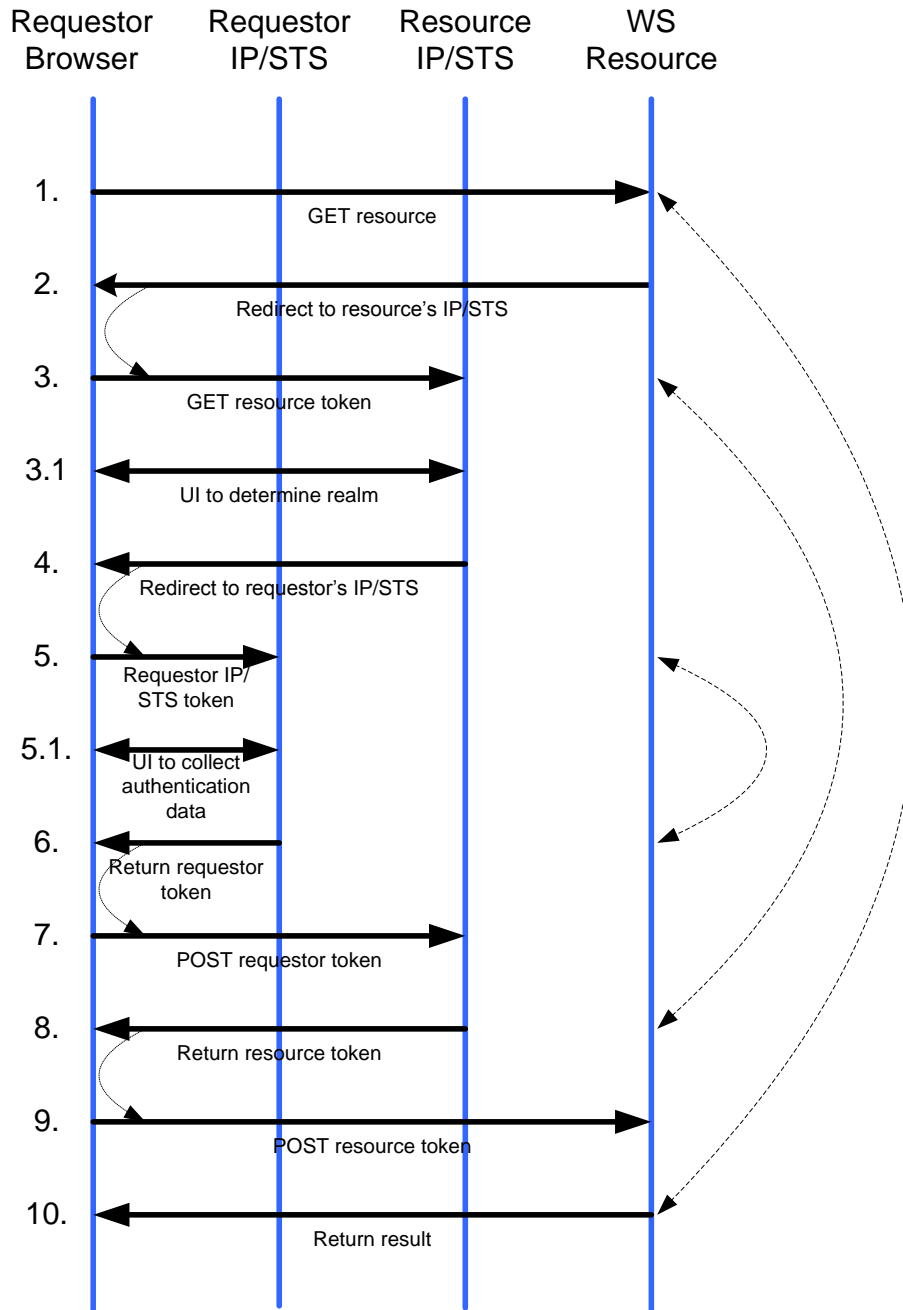
3531 wresultptr

3532 This optional OPTIONAL parameter specifies URL from which the result element can be
3533 obtained.

3534 13.3 Detailed Example of Web Requester Syntax

3535 This section provides a detailed example of the protocol defined in this specification. The exact flow for
3536 Web sign-in scenarios can vary significantly; however, the following diagram and description depict a
3537 *common* or basic sequence of events.

3538 In this scenario, the user at a requestor browser is attempting to access a resource which requires
3539 security authentication to be validated by the resource's security token service. In this example there is a
3540 resource-specific IP/STS.



3541

3542

Figure 29: Details Sample Browser Sign-In

3543

Simple Scenario:

3544

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.

3545

3546

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

3547

Step 2: At the resource, the requestor's request is redirected to the IP/STS associated with the target

3548

resource. The redirected URL MAY contain additional information reflecting agreements which the

3549

resource and its IP/STS have established; however, this (redirection target) URL MUST be used

3550

3551 throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard
3552 HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method
3553 described in step 6).

3554 It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of
3555 the information.

3556 **Step 3:** Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor
3557 realm MAY be cached in an artifact/cookie from an earlier exchange, it MAY be known to or fixed by the
3558 resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

3559 **Step 3.1:** This is an OPTIONAL step. If the resource IP/STS cannot determine the requestor's realm,
3560 then the IP/STS MAY prompt the requestor for realm information.

3561 **Step 4:** The resource IP/STS redirects to the requestor's IP/STS in order to validate the requestor.
3562 Typically, this is done using a HTTP 302 redirect.

3563 As in step 2, additional information MAY be passed to reflect the agreement between the two IP/STS's,
3564 and this request for the token MAY be done using a POST method (see syntax for details).

3565 The requestor IP/STS SHOULD provide information confidentiality or use HTTP/S or some other
3566 transport-level security mechanism.

3567 **Step 5:** The requestor's IP/STS now authenticates the requestor to establish a sign in.

3568 **Step 5.1:** Validation of the requestor MAY involve displaying some UI in this OPTIONAL step.

3569 **Step 6:** Once requestor information has been successfully validated, a security token response (RSTR) is
3570 formatted and sent to the resource IP/STS.

3571 Processing continues at the resource IP/STS via a redirect.

3572 While an IP/STS MAY choose to return a pointer to token information using `wresultptr`, it is
3573 RECOMMENDED that, whenever possible to return the security token (RSTR) using a POST method to
3574 reduce the number of overall messages. This MAY be done using requestor-side scripting. The exact
3575 syntax is described in Appendix I.

3576 **Step 7:** Resource's IP/STS receives and validates the requestor's security token (RSTR).

3577 **Step 8:** The resource's IP/STS performs a federated authentication/authorization check (validation
3578 against policy). After a successful check, the resource's IP/STS can issue a security token for the
3579 resource. The resource IP/STS redirects to the resource.

3580 It should be noted that the OPTIONAL `wctx` parameter specifies the opaque context information (if any)
3581 passed in with the original request and is echoed back here. This mechanism is an optional way for the
3582 IP/STS to have state returned to it.

3583 At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the
3584 requestor (which likely includes the requestor's realm).

3585 **Step 9:** The resource receives the security token (RSTR) from the resource IP/STS. On successful
3586 validation the resource processes the request (per policy).

3587 The security token SHOULD be passed using an HTML POST using the syntax previously described.

3588 **Step 10:** The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it
3589 returns the result of the resource request.

3590

3591 **Optimized Scenario:**

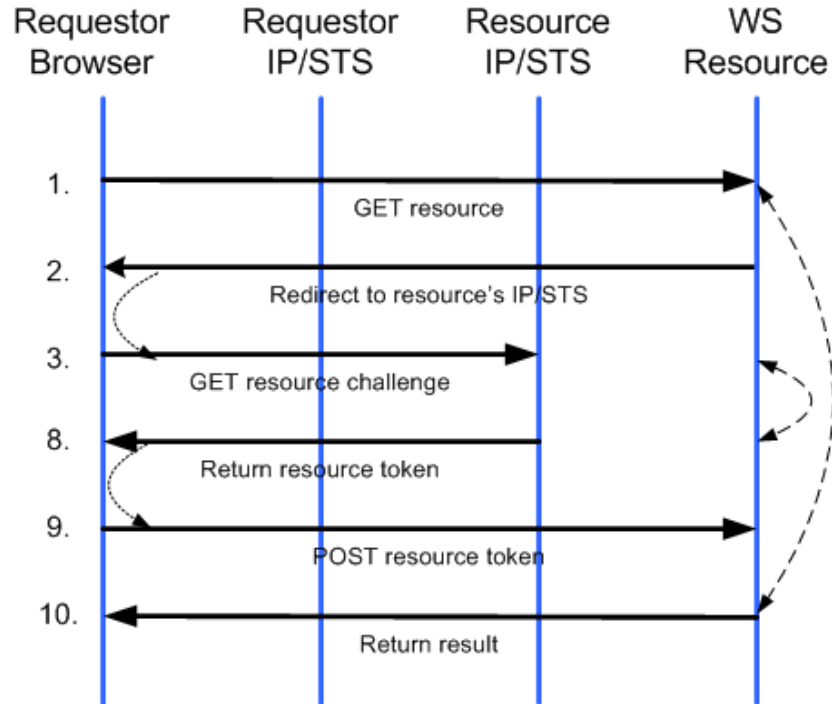


Figure 30: Optimized Sample Browser Sign-In

3592

3593

3594 This scenario assumes that an initial federated flow has occurred. Note that many legs of the initial flow
 3595 MAY be eliminated due to the presence of artifacts/cookies. For readability, the similar steps are
 3596 numbered consistently with the previous non-optimized example.

3597 **Step 1:** The requestor browser accesses a resource, typically using the HTTP GET method.

3598 **Step 2:** At the resource, the requestor's request is redirected to the IP/STS associated with the target
 3599 resource. The redirected URL MAY contain additional information reflecting agreements which the
 3600 resource and its IP/STS have established; however, this (redirection target) URL MUST be used
 3601 throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard
 3602 HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method
 3603 described in step 6).

3604 It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of
 3605 the information.

3606 **Step 3:** Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor
 3607 realm could be cached in an artifact/cookie from an earlier exchange, it could be known to or fixed by the
 3608 resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

3609 **Step 8:** The resource's IP/STS performs a federated authentication/authorization check (validation
 3610 against policy). After a successful check, the resource's IP/STS can issue a security token for the
 3611 resource. The resource IP/STS redirects to the resource.

3612 It should be noted that the OPTIONAL `wctx` parameter specifies the opaque context information (if any)
 3613 passed in with the original request and is echoed back here. This mechanism is an optional way for the
 3614 IP/STS to have state returned to it.

3615 At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the
 3616 requestor (which likely includes the requestor's realm).

3617 **Step 9:** The resource receives the security token (RSTR) from the resource IP/STS. On successful
3618 validation the resource processes the request (per policy).
3619 The security token SHOULD be passed using an HTML POST using the syntax previously described.
3620 **Step 10:** The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it
3621 returns the result of the resource request.

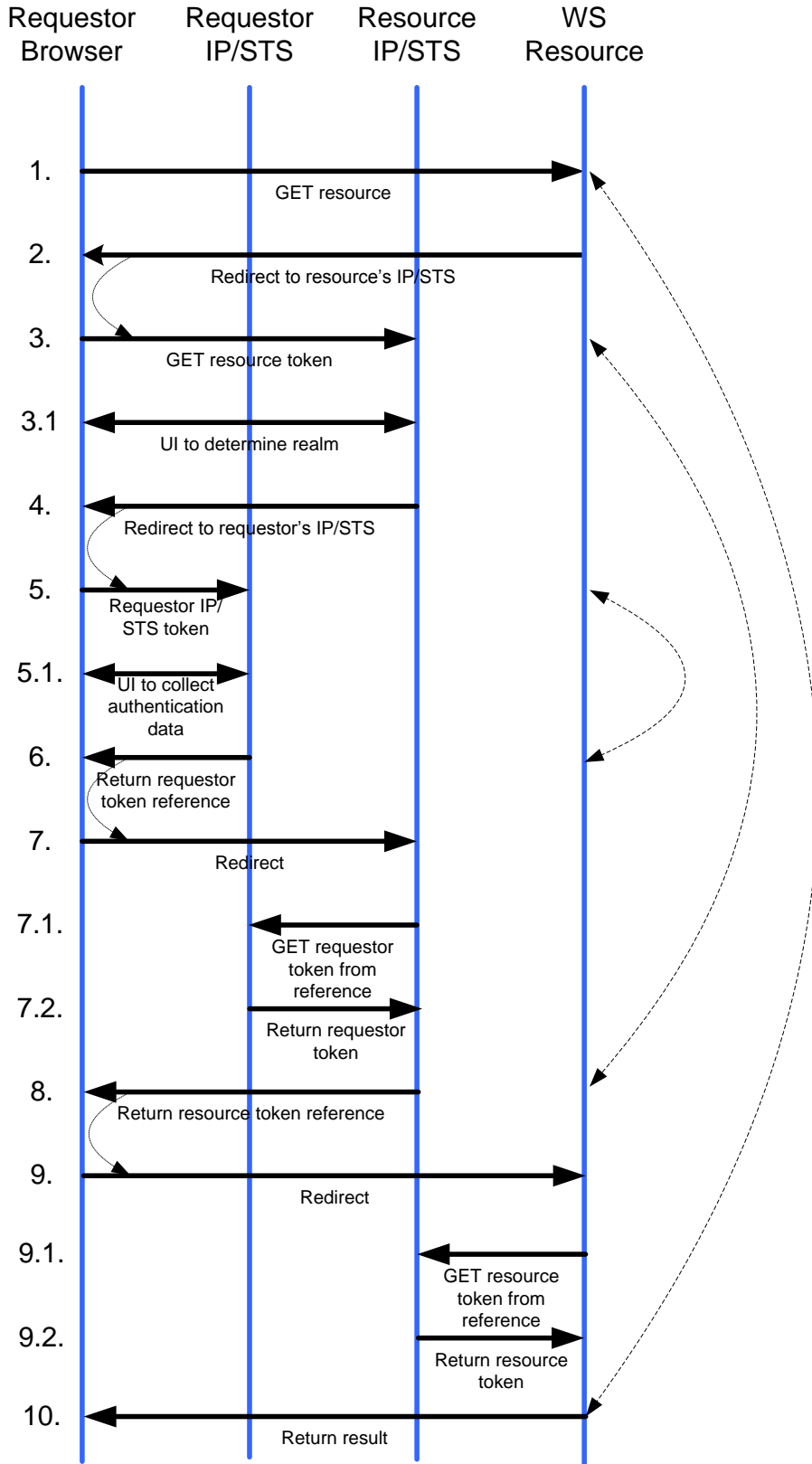
3622 **13.4 Request and Result References**

3623 The previous example illustrates a common form of messaging when passing WS-Trust messages via a
3624 simple Web browser. However, in some scenarios it is undesirable to use POST messages and carry the
3625 full details within the messages (e.g. when redirecting through wireless or mobile devices). In such cases
3626 requests and responses can be referenced via a URL and all messages passed as part of the query
3627 strings (or inside small POSTs).

3628 Request references are specified via *wreqptr* and typically specify a `<wst:RequestSecurityToken>`
3629 element that can be obtained by issuing a HTTP GET against the specified URL. Response references
3630 are specified via *wresultptr* and typically specify a `<wst:RequestSecurityTokenResponse>` or
3631 `<wst:RequestSecurityTokenResponseCollection>` element that can be obtained by issuing a
3632 HTTP GET against the specified URL.

3633 This section provides a detailed example of the use of references with the protocol defined in this
3634 specification. The exact flow for Web sign-in scenarios can vary significantly; however, the following
3635 diagram and description depict a *common* or basic sequence of events. Note that this example only
3636 illustrates result reference not request references and makes use of a resource-specific IP/STS.

3637 In this scenario, the user at a requestor browser is attempting to access a resource which requires
3638 security authentication to be validated by the resource's security token service.



3639

3640

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

3641 **Step 1:** The requestor browser accesses a resource, typically using the HTTP GET method.

3642 **Step 2:** At the resource, the requestor's request is redirected to the IP/STS associated with the target
3643 resource. The redirected URL MAY contain additional information reflecting agreements which the
3644 resource and its IP/STS have established; however, this (redirection target) URL MUST be used
3645 throughout the protocol as the URL for the resource's IP/STS. Typically, this occurs using a standard
3646 HTTP 302 error code. (Alternatively, the request for the token MAY be done using a HTTP POST method
3647 described in step 6).

3648 It is RECOMMENDED that the resource STS provide confidentiality (e.g. using encryption or HTTP/S) of
3649 the information.

3650 **Step 3:** Upon receipt of the redirection, the IP/STS must determine the requestor realm. This requestor
3651 realm could be cached in an artifact/cookie from an earlier exchange, it could be known to or fixed by the
3652 resource, or the requestor MAY be prompted to enter or select their realm (step 3.1).

3653 **Step 3.1:** This is an OPTIONAL step. If the resource IP/STS cannot determine the requestor's realm,
3654 then the IP/STS MAY prompt the requestor for realm information.

3655 **Step 4:** The resource IP/STS redirects to the requestor's IP/STS in order to validate the requestor.
3656 Typically, this is done using a HTTP 302 redirect.

3657 As in step 2, additional information MAY be passed to reflect the agreement between the two IP/STS's,
3658 and this request for the token MAY be done using a POST method (see syntax for details).

3659 The requestor IP/STS SHOULD provide information confidentiality or use HTTP/S or some other
3660 transport-level security mechanism.

3661 **Step 5:** The requestor's IP/STS now authenticates the requestor to establish a sign in.

3662 **Step 5.1:** Validation of the requestor MAY involve displaying some UI in this OPTIONAL step.

3663 **Step 6:** Once requestor information has been successfully validated, a security token response (RSTR) is
3664 formatted and sent to the resource IP/STS.

3665 Processing continues at the resource IP/STS via a redirect.

3666 **Step 7:** Resource's IP/STS receives and validates the requestor's security token (RSTR).

3667 **Step 7.1:** The Resource IP/STS issues a GET to the Requestor IP/STS to obtain the actual RSTR.

3668 **Step 7.2:** The Requestor IP/STS responds to the GET and returns the actual RSTR.

3669 **Step 8:** The resource's IP/STS performs a federated authentication/authorization check (validation
3670 against policy). After a successful check, the resource's IP/STS can issue a security token for the
3671 resource. The resource IP/STS redirects to the resource.

3672 It should be noted that the OPTIONAL `wctx` parameter specifies the opaque context information (if any)
3673 passed in with the original request and is echoed back here. This mechanism is an optional way for the
3674 IP/STS to have state returned to it.

3675 At this point the resource's IP/STS MAY choose to set an artifact/cookie to indicate the sign-in state of the
3676 requestor (which likely includes the requestor's realm).

3677 **Step 9:** The resource receives the security token (RSTR) from the resource IP/STS. On successful
3678 validation the resource processes the request (per policy).

3679 The security token SHOULD be passed using an HTML POST using the syntax previously described.

3680 **Step 9.1:** The Resource issues a GET to the Resource IP/STS to obtain the actual RSTR.

3681 **Step 9.2:** The Resource IP/STS responds to the GET and returns the actual RSTR.

3682 **Step 10:** The resource MAY establish a artifact/cookie indicating the sign-in state of the requestor when it
3683 returns the result of the resource request.

3684 **13.5 Home Realm Discovery**

3685 In the protocol previously described the resource or the resource's IP/STS must determine the IP/STS for
3686 the requestor and re-direct to obtain an identity token. After this is done, the information can be cached in
3687 a cookie (or by whatever means is desired).

3688 There is no normative way of discovering the *home realm* of the requestor, however, the following
3689 mechanisms are common methods:

- 3690 • *Fixed* – The home realm is fixed or known
- 3691 • *Requestor IP* – The home realm is determined using the requestor's IP address
- 3692 • *Prompt* – The user is prompted (typically using a Web page)
- 3693 • *Discovery Service* – A service is used to determine the home realm
- 3694 • *Shared Cookie* – A shared cookie from a shared domain is used (out of scope)

3695 The first three mechanisms are well understood, the *Discovery Service* is discussed next, and the cookie
3696 mechanism is outside the scope of this document.

3697 **13.5.1 Discovery Service**

3698 The *Home Realm Discovery Service* is a Web-based service that, through implementation-specific
3699 methods MAY be able to determine a requestor's home realm without user interaction.

3700 A resource or resource IP/STS MAY redirect to a discovery service to attempt to determine the home
3701 realm without prompting the user. The discovery service MUST redirect back to the URL specified by the
3702 *wreply* parameter. If the context parameter is specified it MUST also be specified. If the discovery
3703 service was able to determine the home realm, it is returned using the *whr* parameter defined in section
3704 13.2.2. This parameter contains a URI which identifies the home realm of the user. This SHOULD be the
3705 same URI that the user's realm uses for the *wrealm* parameter when it makes token requests to other
3706 federated partners. This value can be used to lookup the URL for the user's IP/STS for properly
3707 redirecting the token request.

3708 If the discovery service is unable to determine the home realm then the *whr* parameter is not specified
3709 and the home realm must be discovered by other means.

3710 **13.6 Minimum Requirements**

3711 For the purposes of interoperability of federated Web Single Sign-on, this sub-section defines a subset of
3712 the exchanges defined in this chapter which MUST be supported by all Web-enabled requestors and
3713 services. Optional aspects are optional for both clients and services.

3714 The scenario and diagram(s) in section 13.3 illustrates the core Sign-On messages between two
3715 federated realms. This is the center of the interoperability subset described below.

3716 **13.6.1 Requesting Security Tokens**

3717 The focus of these requirements is on the message exchange between the requestor IP/STS and the
3718 resource IP/STS. Thus, to conform to this specification, messages 1, 4, 7 & 10 MUST be supported
3719 (again refer to the figure and steps in section 13.3). All other message exchanges are implementation
3720 specific and are only provided here for guidance.

3721 A security token is requested via SignIn message in step 2 of the diagram. Message 3 arrives via HTTP
3722 GET and is protected by SSL/TLS. The parameters are encoded in a query string as specified in section
3723 13.2. The message will contain parameters as detailed below. Parameters enclosed in brackets are
3724 OPTIONAL.

3725

3726
3727
3728
3729
3730

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

3731

3732 The REQUIRED *wa* field is common to all SignIn messages and is fixed.

3733 The REQUIRED *wrealm* field MUST contain a URI that the *Resource IP/STS* and *Requestor IP/STS*
3734 have agreed to use to identify the realm of *Resource IP/STS* in messages to *Requestor IP/STS*.

3735 The OPTIONAL *wreply* field specifies the URL to which this message's response will be POSTed (see
3736 Returning Security Tokens).

3737 The OPTIONAL *wctx* field is provided for *Resource IP/STS*'s use and MUST be returned by *Requestor*
3738 *IP/STS* unchanged.

3739 The OPTIONAL *wct* field, if present, MUST contain the current time in UTC using the ISO8601 format
3740 (e.g. "2003-04-30T22:47:20Z"). This field MAY not be available if the requestor is coming via a portal link.
3741 Individual implementations of *Requestor IP/STS* MAY require this field to be present.

3742 Other options MAY be specified but are not required to be supported.

3743 13.6.2 Returning Security Tokens

3744 A security token is returned in response to successful Web SignIn messages, as described in the
3745 example protocol message flow in section 13.3. Security tokens are returned to the requestor and
3746 SHOULD be transmitted to a Resource Provider via HTTP POST and be protected by SSL/TLS, as
3747 depicted in steps 6-7 and 9-10 of figure 29. Optionally, the token MAY be returned using the *wresultptr*
3748 parameter. Encoding of the parameters in the POST body MUST be supported. The parameters to the
3749 message MAY be encoded in the query string if *wresultptr* is being used. The message will contain
3750 parameters as detailed below. Parameters enclosed in brackets are OPTIONAL.

3751

3752
3753
3754
3755

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

3756

3757 The REQUIRED *wa* field is common to all SignIn messages and is fixed.

3758 The REQUIRED *wresult* field MUST contain a `<wst:RequestSecurityTokenResponse>` element, as
3759 detailed below.

3760 The OPTIONAL *wctx* field MUST be identical to the *wctx* field from the incoming SignIn message that
3761 evoked this response.

3762 The OPTIONAL *wresultptr* field provides a pointer to the resulting
3763 `<wst:RequestSecurityTokenResponse>` element, as detailed below.

3764 13.6.3 Details of the RequestSecurityTokenResponse element

3765 The `<wst:RequestSecurityTokenResponse>` element that is included as the *wresult* field in the
3766 SignIn response MUST contain a `<wst:RequestedSecurityToken>` element. Support for SAML
3767 assertions MUST be provided but another token format MAY be used (depending on policy).

3768 The `<wst:RequestSecurityTokenResponse>` element MAY include a *wsp:AppliesTo* /
3769 *wsa:EndpointReference* / *wsa:Address* element that specifies the Resource Realm URI. Note that
3770 this data MUST be consistent with similar data present in security tokens (if any is present) – for example

3771 it must duplicate the information in the signed token's *saml:Audience* element when SAML security
3772 tokens are returned.

3773 **13.6.4 Details of the Returned Security Token Signature**

3774 It MUST be possible to return signed security tokens, but unsecured tokens MAY be returned. Signed
3775 security tokens SHOULD contain an enveloped signature to prevent tampering but MAY use alternative
3776 methods if the security token format allows for specialized augmentation of the token. The signature
3777 SHOULD be performed over canonicalized XML [XML-C14N] (failure to do so MAY result in non-verifiable
3778 security tokens). The signature SHOULD be produced using the *Requestor STS* private key, which
3779 SHOULD correspond to either a security token included as part of the response or pre-established with
3780 the requestor. Note that in the above example the certificate is included directly in KeyInfo (via the
3781 X509Data element [WSS:X509Token]). This is the RECOMMENDED approach.

3782 When used, the X509SKI element contains the base64 encoded plain (i.e., non-DER-encoded) value of
3783 an X509 V.3 SubjectKeyIdentifier extension. If the SubjectKeyIdentifier field is not present in the
3784 certificate, the certificate itself MUST be included directly in KeyInfo (see the above example).

3785 Note that typically the returned security token is unencrypted (The entire RSTR is sent over SSL3.0/TLS
3786 [HTTPS]) but it MAY be encrypted in specialized scenarios.

3787 Take care to include appropriate transforms in *Signature/Reference/Transforms*. For example, all SAML
3788 tokens [WSS:SAMLTokenProfile] following the rules above MUST contain the enveloped signature and
3789 EXCLUSIVE canonicalization transforms.

3790 **13.6.5 Request and Response References**

3791 If the *wreqptr* or *wresultptr* parameters are supported, it MUST be possible to pass
3792 `<wst:RequestSecurityToken>` in the *wreqptr* and either
3793 `<wst:RequestSecurityTokenResponse>` or
3794 `<wst:RequestSecurityTokenResponseCollection>` in *wresultptr*. Other values MAY (but are not
3795 required) to be supported.

3796 14 Additional Policy Assertions

3797 This specification defines the following assertions for use with [WS-Policy] and [WS-SecurityPolicy].

3798 14.1 RequireReferenceToken Assertion

3799 This element represents a requirement to include a ReferenceToken (as described previously in this
3800 specification). The default version of this token is the version described in this document.

3801 The syntax is as follows:

```
3802 <fed:RequireReferenceToken sp:IncludeToken="xs:anyURI" ? ... >  
3803 <wsp:Policy>  
3804 <fed:RequireReferenceToken11 ...>...</fed:RequireReferenceToken11 > ?  
3805 ...  
3806 </wsp:Policy> ?  
3807 ...  
3808 </fed:RequireReferenceToken>
```

3809 The following describes the attributes and elements listed in the schema outlined above:

3810 /fed:RequireReferenceToken

3811 This identifies a RequireReference assertion

3812 /fed:RequireReferenceToken/sp:IncludeToken

3813 This OPTIONAL attribute identifies the token inclusion value for this token assertion

3814 /fed:RequireReferenceToken/wsp:Policy

3815 This OPTIONAL element identifies additional requirements for use of the
3816 fed:RequireReferenceToken assertion.

3817 /fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11

3818 This OPTIONAL element indicates that a reference token should be used as defined in this
3819 specification.

3820 /fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11/@{any}

3821 This extensibility mechanism allows attributes to be added. Use of this extensibility point MUST
3822 NOT violate or alter the semantics defined in this specification.

3823 /fed:RequireReferenceToken/wsp:Policy/fed:RequireReferenceToken11/{any}

3824 This is an extensibility point allowing content elements to be specified. Use of this extensibility
3825 point MUST NOT alter semantic defined in this specification.

3826 /fed:RequireReferenceToken/@{any}

3827 This extensibility mechanism allows attributes to be added . Use of this extensibility point MUST
3828 NOT violate or alter the semantics defined in this specification.

3829 /fed:RequireReferenceToken/{any}

3830 This is an extensibility point allowing content elements to be specified. Use of this extensibility
3831 point MUST NOT alter semantic defined in this specification.

3832 This assertion is used wherever acceptable token types are identified (e.g. within the supporting token
3833 assertions defined in WS-SecurityPolicy).

3834 14.2 WebBinding Assertion

3835 The WebBinding assertion is used in scenarios where requests are made of token services using a Web
3836 client and HTTP with GET, POST, and redirection as described in Section 13. Specifically, this assertion
3837 indicates that the requests use the Web client mechanism defined in this document and are protected
3838 using the means provided by a transport. This binding has several specific binding properties:

- 3839 • The [TransportToken] property indicates what transport mechanism is used to protect requests
3840 and responses.
- 3841 • The [AuthenticationToken] property indicates the REQUIRED token type for authentication. Note
3842 that this can be a choice of formats as it uses nested policy. Also note that this can specify
3843 fed:ReferenceToken as an option to indicate that token handles are accepted (and dereferenced).
- 3844 • The [RequireSignedTokens] property indicates that tokens MUST be signed i.e. only tokens that
3845 are signed are accepted.
- 3846 • The [RequireBearerTokens] property indicates that tokens MUST be bearer tokens i.e only
3847 bearer tokens are accepted.
- 3848 • The [RequireSharedCookies] property indicates if shared cookies MUST be used for home realm
3849 discovery
- 3850 • The [Base64Url] property indicates that base64url encoded xml parameter content is REQUIRED.

3851 The syntax is as follows:

```
3852 <fed:WebBinding ...>  
3853   <wsp:Policy>  
3854     <sp:TransportToken ...> ... </sp:TransportToken> ?  
3855     <fed:AuthenticationToken ... > ?  
3856       <wsp:Policy> ... </wsp:Policy>  
3857       <fed:ReferenceToken ...>... </fed:ReferenceToken> ?  
3858     </fed:AuthenticationToken>   <fed:RequireSignedTokens ... /> ?  
3859     <fed:RequireBearerTokens ... /> ?  
3860     <fed:RequireSharedCookies ... /> ?  
3861     <fed:Base64Url ... /> ?  
3862     ...  
3863   </wsp:Policy> ?  
3864 </fed:WebBinding>
```

3865 The following describes the attributes and elements listed in the schema outlined above:

3866 /fed:WebBinding

3867 This identifies a WebBinding assertion

3868 /fed:WebBinding/wsp:Policy

3869 This identifies a nested `wsp:Policy` element that defines the behavior of the WebBinding
3870 assertion.

3871 /fed:WebBinding/wsp:Policy/sp:TransportToken

3872 This indicates that a Transport Token as defined in [WS-SecurityPolicy] is REQUIRED

3873 /fed:WebBinding/wsp:Policy/fed:AuthenticationToken

3874 This indicates the REQUIRED token type for authentication.

3875 /fed:WebBinding/wsp:Policy/fed:AuthenticationToken/wsp:Policy

3876 This indicates a nested `wsp:Policy` element to specify a choice of formats for the authentication
3877 token.

3878 /fed:WebBinding/wsp:Policy/fed:AuthenticationToken/fed:ReferenceToken

- 3879 This OPTIONAL element indicates token handles that are accepted. See section 8.1 for a
3880 complete description.
- 3881 /fed:WebBinding/wsp:Policy/RequireSignedTokens
- 3882 This indicates a requirement for tokens to be signed. This sets the [RequireSignedTokens]
3883 property to true (the default value is false).
- 3884 /fed:WebBinding/wsp:Policy/RequireBearerTokens
- 3885 This indicates a requirement for bearer tokens. This sets the [RequireBearerTokens] property to
3886 *true* (the default value is *false*).
- 3887 /fed:WebBinding/wsp:Policy/RequireSharedCookies
- 3888 This indicates a requirement for shared cookies to facilitate home realm discovery. This sets the
3889 [RequireSharedCookies] property to *true* (the default value is *false*).
- 3890 /fed:WebBinding/wsp:Policy/Base64Url
- 3891 This indicates a requirement for xml parameter content to be base64url encoded. This sets the
3892 [Bas64Url] property to true (the default value is false).
- 3893 Note that the `sp:AlgorithmSuite`, `sp:Layout`, and `sp:IncludeTimestamp` properties are not used
3894 by this binding and SHOULD NOT be specified.
- 3895 This assertion SHOULD only be used with endpoint subjects.

3896 14.3 Authorization Policy

- 3897 To indicate support for the authorization features described in this specification, the following policy
3898 assertions are specified.

```
3899 <fed:RequiresGenericClaimDialect ... />
3900 <fed:IssuesSpecificMetadataFault ... />
3901 <fed:AdditionalContextProcessed ... />
```

- 3902 The following describes the above syntax:
- 3903 /fed:RequiresGenericClaimDialect
- 3904 This assertion indicates that the use of the generic claim dialect defined in this specification in
3905 Section 9.3 is REQUIRED by the service.
- 3906 /fed:IssuesSpecificPolicyFault
- 3907 This assertion indicates that the service issues the `fed:SpecificPolicy` Fault defined in this
3908 document if the security requirements for a specific request are beyond those of the base policy.
- 3909 /fed:AdditionalContextProcessed
- 3910 This assertion indicates that the service will process the `fed:AdditionalContext` parameter if
3911 specified in an RST request.
- 3912 Typically these assertions are specified at the service or port/endpoint.
- 3913 These assertions SHOULD be specified within a binding assertion.

3914

15 Error Handling

3915

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

3916

These errors use the SOAP Fault mechanism. Note that the reason text provided below is

3917

RECOMMENDED, but alternative text MAY be provided if more descriptive or preferred by the

3918

implementation. The table below is defined in terms of SOAP 1.1. For SOAP 1.2 the Fault/Code/Value is

3919

env:Sender (as defined in SOAP 1.2) and the Fault/Code/SubCode/Value is the *faultcode* below, and the

3920

Fault/Reason/Text is the *faultstring* below. It should be noted that profiles MAY provide second-level

3921

detail fields but they should be careful not to introduce security vulnerabilities when doing so (e.g. by

3922

providing too detailed information or echoing confidential information over insecure channels). It is

3923

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

3924

16 Security Considerations

3925 It is strongly RECOMMENDED that the communication between services be secured using the
3926 mechanisms described in [WS-Security]. In order to properly secure messages, the body and all relevant
3927 headers need to be included in the signature.

3928 Metadata that is exchanged also needs to be secured to prevent various attacks. All metadata
3929 documents SHOULD be verified to ensure that the issuer can speak for the specified endpoint and that
3930 the metadata is what the issuer intended.

3931 All federation-related messages such as sign-out, principal, attribute, and pseudonym management
3932 SHOULD be integrity protected (signed or use transport security). If a message is received where the
3933 body is not integrity protected, it is RECOMMENDED that the message not be processed.

3934 All sign-out requests SHOULD be signed by the principal being purported to be signing in or out, or by a
3935 principal that is authorized to be on behalf of the indicated principal.

3936 It is also RECOMMENDED that all messages be signed by the appropriate security token service. If a
3937 message is received that does not have a signature from a principal authorized to speak for the security
3938 token service, it is RECOMMENDED that the message not be processed.

3939 When using Web messages care should be taken around processing of the *wreply* parameter as its value
3940 could be spoofed. It is RECOMMENDED that implementations do explicit lookup and verification of URL,
3941 and that these values be passed with transport security.

3942 The attribute service maintains information that may be very sensitive. Significant care SHOULD be
3943 taken to ensure that a principal's privacy is taken into account first and foremost.

3944 The pseudonym service may contain passwords or other information used in proof-of-possession
3945 mechanisms. Extreme care needs to be taken with this data to ensure that it cannot be compromised. It
3946 is strongly RECOMMENDED that such information be encrypted over communications channels and in
3947 any physical storage.

3948 If a security token does not contain an embedded signature (or similar integrity mechanism to protect
3949 itself), it SHOULD be included in any message integrity mechanisms (e.g. included in the message
3950 signature).

3951 If privacy is a concern, the security tokens used to authenticate and authorize messages MAY be
3952 encrypted for the authorized recipient(s) using mechanisms in WS-Security.

3953 Care SHOULD be taken when processing and responding to requests from 3rd-parties to mitigate
3954 potential information disclosure attacks by way of faulting requests for specific claims.

3955 As a general rule tokens SHOULD NOT have lifetimes beyond the minimum of the basis credentials
3956 (security tokens). However, in some cases special arrangements may exist and issuers may provide
3957 longer lived tokens. Care SHOULD be taken in such cases not to introduce security vulnerabilities.

3958 The following list summarizes common classes of attacks that apply to this protocol and identifies the
3959 mechanism to prevent/mitigate the attacks. Note that wherever WS-Security is suggested as the
3960 mitigation, [HTTPS] is the corresponding mechanism for Web requestors:

- 3961 • **Metadata alteration** – Alteration is prevented by including signatures in metadata or using secure
3962 channels for metadata transfer.
- 3963 • **Message alteration** – Alteration is prevented by including signatures of the message information
3964 using [WS-Security].
- 3965 • **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using [WS-Security].
- 3966 • **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by comparing
3967 secured policies – see [WS-Policy] and [WS-SecurityPolicy]).

- 3968 • **Authentication** – Authentication is established using the mechanisms described in [WS-Security]
3969 and [WS-Trust]. Each message is authenticated using the mechanisms described in [WS-Security].
- 3970 • **Accountability** – Accountability is a function of the type of and string of the key and algorithms being
3971 used. In many cases, a strong symmetric key provides sufficient accountability. However, in some
3972 environments, strong PKI signatures are required.
- 3973 • **Availability** – All reliable messaging services are subject to a variety of availability attacks. Replay
3974 detection is a common attack and it is RECOMMENDED that this be addressed by the mechanisms
3975 described in [WS-Security]. Other attacks, such as network-level denial of service attacks are harder
3976 to avoid and are outside the scope of this specification. That said, care SHOULD be taken to ensure
3977 that minimal state is saved prior to any authenticating sequences.
- 3978 • **Replay attacks:** It is possible that requests for security tokens could be replayed. Consequently, it
3979 is RECOMMENDED that all communication between Security Token Services and resources take
3980 place over secure connections. All cookies indicating state SHOULD be set as secure.
- 3981 • **Forged security tokens:** Security token services MUST guard their signature keys to prevent
3982 forging of tokens and requestor identities.
- 3983 • **Privacy:** Security token services SHOULD NOT send requestors' personal identifying information or
3984 information without getting consent from the requestor. For example a Web site SHOULD NOT
3985 receive requestors' personal information without an appropriate consent process.
- 3986 • **Compromised services:** If a Security Token Service is compromised, all requestor accounts
3987 serviced SHOULD be assumed to be compromised as well (since an attacker can issue security
3988 tokens for any account they want). However they SHOULD NOT not be able to issue tokens directly
3989 for identities outside the compromised realm. This is of special concern in scenarios like the 3rd party
3990 brokered trust where a 3rd party IP/STS is brokering trust between two realms. In such a case
3991 compromising the broker results in the ability to indirectly issue tokens for another realm by indicating
3992 trust.
- 3993 As with all communications careful analysis SHOULD be performed on the messages and interactions to
3994 ensure they meet the desired security requirements.
3995

3996

17 Conformance

- 3997 An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level
3998 requirements defined within this specification. A SOAP Node MUST NOT use the XML namespace
3999 identifier for this specification (listed in Section 1.4) within SOAP Envelopes unless it is compliant with this
4000 specification.
- 4001 This specification references a number of other specifications (see the table above). In order to comply
4002 with this specification, an implementation MUST implement the portions of referenced specifications
4003 necessary to comply with the required provisions of this specification. Additionally, the implementation of
4004 the portions of the referenced specifications that are specifically cited in this specification MUST comply
4005 with the rules for those portions as established in the referenced specification.
- 4006 Additionally normative text within this specification takes precedence over normative outlines (as
4007 described in section 1.3), which in turn take precedence over the XML Schema [XML Schema Part 1,
4008 Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further
4009 constrains the schemas and/or WSDL that are part of this specification; and this specification contains
4010 further constraints on the elements defined in referenced schemas.
- 4011 If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for
4012 any other unrecognized/unsupported message. If an OPTIONAL message is supported, then the
4013 implementation MUST satisfy all of the MUST and REQUIRED sections of the message.

4014 Appendix A WSDL

4015 The following illustrates the WSDL for the Web service methods described in this specification:

```
4016 <wsdl:definitions xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
4017     xmlns:xs='http://www.w3.org/2001/XMLSchema'
4018     xmlns:tns='http://docs.oasis-open.org/wsfed/federation/200706'
4019     targetNamespace='http://docs.oasis-open.org/wsfed/federation/200706' >
4020
4021 <!-- WS-Federation endpoints implement WS-Trust -->
4022 <wsdl:import namespace='http://docs.oasis-open.org/ws-sx/ws-trust/200512'
4023     location='http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3.wsdl'
4024 />
4025
4026 <!-- WS-Federation endpoints can implement WS-MEX -->
4027 <wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/09/mex'
4028     location='http://schemas.xmlsoap.org/ws/2004/09/mex/MetadataExchange.wsdl' />
4029
4030 <!-- WS-Federation endpoints can implement WS-Eventing -->
4031 <wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/08/eventing'
4032     location='http://schemas.xmlsoap.org/ws/2004/08/eventing/eventing.wsdl' />
4033
4034 <!-- WS-Federation endpoints can implement WS-Transfer -->
4035 <wsdl:import namespace='http://schemas.xmlsoap.org/ws/2004/09/transfer'
4036     location='http://schemas.xmlsoap.org/ws/2004/09/transfer/transfer.wsdl' />
4037
4038 <!-- WS-Federation endpoints can implement WS-ResourceTransfer -->
4039 <wsdl:import
4040     namespace='http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer'
4041     location='http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/wsrt.wsdl' />
4042
4043 <wsdl:types>
4044 <xs:schema
4045     <xs:import namespace='http://docs.oasis-open.org/wsfed/federation/200706' />
4046 </xs:schema>
4047 </wsdl:types>
4048
4049 <wsdl:message name='SignOut' >
4050 <wsdl:part name='Body' element='tns:SignOut' />
4051 </wsdl:message>
4052
4053 <wsdl:portType name='SignOutIn' >
4054 <wsdl:operation name='SignOut' >
4055 <wsdl:input message='tns:SignOut' />
4056 </wsdl:operation>
4057 </wsdl:portType>
4058
4059 <wsdl:portType name='SignOutOut' >
4060 <wsdl:operation name='SignOut' >
4061 <wsdl:output message='tns:SignOut' />
4062 </wsdl:operation>
4063 </wsdl:portType>
4064
4065 </wsdl:definitions>
```

4066
4067
4068
4069
4070

Appendix B Sample HTTP Flows for Web Requestor Detailed Example

This appendix provides sample HTTP messages for the detailed example previously described in the Web requestor section.

In this example, the following URLs are used:

<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

4071

Step 1 – GET resource

4072

```
GET https://res.resource.com/sales HTTP/1.1
```

4073

Step 2 – Redirect to resource's IP/STS

4074

```
HTTP/1.1 302 Found ↓
```

4075

```
Location:
```

4076

```
https://sts.resource.com/sts?wa=wsignin1.0&wreply=https://res.resource.com/sales&wct=2003-03-03T19:06:21Z
```

4077

4078

In addition, the resource could check for a previously written artifact/cookie and, if present, skip to Step 10.

4079

4080

Step 3 – GET resource challenge

4081

```
GET https://sts.resource.com/sts?wa=wsignin1.0&wreply=
```

4082

```
https://res.resource.com/sales&wct=2003-03-03T19:06:21Z HTTP/1.1
```

4083

Step 3.1 – UI to determine realm (OPTIONAL)

4084

```
[Implementation Specific Traffic]
```

4085

Step 4 – Redirect to requestor's IP/STS

4086

```
HTTP/1.1 302 Found ↓
```

4087

```
Location: https://sts.account.com/sts?wa=wsignin1.0&wreply=
```

4088

```
https://sts.resource.com/sts&wctx= https://res.resource.com/sales&wct=2003-03-03T19:06:22Z&wtrealm=resource.com
```

4089

4090

In addition, the Resource IP/STS MAY check for a previously written artifact/cookie and, if present, skip to Step 8.

4091

4092

Step 5 – Requestor IP/STS challenge

4093

```
GET
```

4094

```
https://sts.account.com/sts?wa=wsignin1.0&wreply=https://sts.resource.com/sts&wctx=https://res.resource.com/sales&wct=2003-03-
```

4095

```
03T19:06:22Z&wtrealm=resource.com HTTP/1.1
```

4096

4097

Step 5.1 – UI to collect authentication data (OPTIONAL)

4098

[Implementation Specific Traffic]

4099

Step 6 – Return requestor token

4100

```
HTTP/1.1 200 OK
```

4101

```
...
```

4102

4103

```
<html xmlns="https://www.w3.org/1999/xhtml">
```

4104

```
<head>
```

4105

```
<title>Working...</title>
```

4106

```
</head>
```

4107

```
<body>
```

4108

```
<form method="post" action="https://sts.resource.com/sts">
```

4109

```
<p>
```

4110

```
<input type="hidden" name="wa" value="wsignin1.0" />
```

4111

```
<input type="hidden" name="wctx" value="https://res.resource.com/sales" />
```

4112

```
<input type="hidden" name="wresult"
```

4113

```
value="&lt;RequestSecurityTokenResponse&gt;...&lt;/RequestSecurityTokenResponse
```

4114

```
e&gt;" />
```

4115

```
<button type="submit">POST</button> <!-- included for requestors that do not
```

4116

```
support javascript -->
```

4117

```
</p>
```

4118

```
</form>
```

4119

```
<script type="text/javascript">
```

4120

```
setTimeout('document.forms[0].submit()', 0);
```

4121

```
</script>
```

4122

```
</body>
```

4123

```
</html>
```

4124

Step 7 – POST requestor token

4125

```
POST https://sts.resource.com/sts HTTP/1.1 ↵
```

4126

```
... ↵
```

4127

```
↵
```

4128

```
wa=wsignin1.0 ↵
```

4129

```
wctx=https://res.resource.com/sales
```

4130

```
wresult=<RequestSecurityTokenResponse>...</RequestSecurityTokenResponse>
```

4131

Step 8 – Return resource token

4132

```
HTTP/1.1 200 OK
```

4133

```
...
```

4134

4135

```
<html xmlns="https://www.w3.org/1999/xhtml">
```

4136

```
<head>
```

4137

```
<title>Working...</title>
```

4138

```
</head>
```

4139

```
<body>
```

4140

```
<form method="post" action="https://res.resource.com/sales">
```

4141

```
<p>
```

4142

```
<input type="hidden" name="wa" value="wsignin1.0" />
```

4143

```
<input type="hidden" name="wresult"
```

4144

```
value="&lt;RequestSecurityTokenResponse&gt;...&lt;/RequestSecurityTokenResponse
```

4145

```
e&gt;" />
```

4146

```
<button type="submit">POST</button> <!-- included for requestors that do not
```

4147

```
support javascript -->
```

4148

```
</p>
```

4149

```
</form>
```

4150

```
<script type="text/javascript">
```

4151

```
setTimeout('document.forms[0].submit()', 0);
```

4152

```
</script>
```

4153

```
</body>
```

4154

```
</html>
```

4155 **Step 9 – POST Resource token**

```
4156 POST https://res.resource.com/sales HTTP/1.1 ↵  
4157 ... ↵  
4158 ↵  
4159 wa=wsignin1.0 ↵  
4160 wresult=<RequestSecurityTokenResponse>...</RequestSecurityTokenResponse>
```

4161 **Step 10 – Return result**

```
4162 [Implementation Specific Traffic]
```

4163

Appendix C Sample Use Cases

4164

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.

4165

4166

C.1 Single Sign On

4167

Requestors use the mechanisms defined within [WS-Security], [WS-Trust], and [WS-Federation] to effect single sign-on.

4171

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.

4172

4173

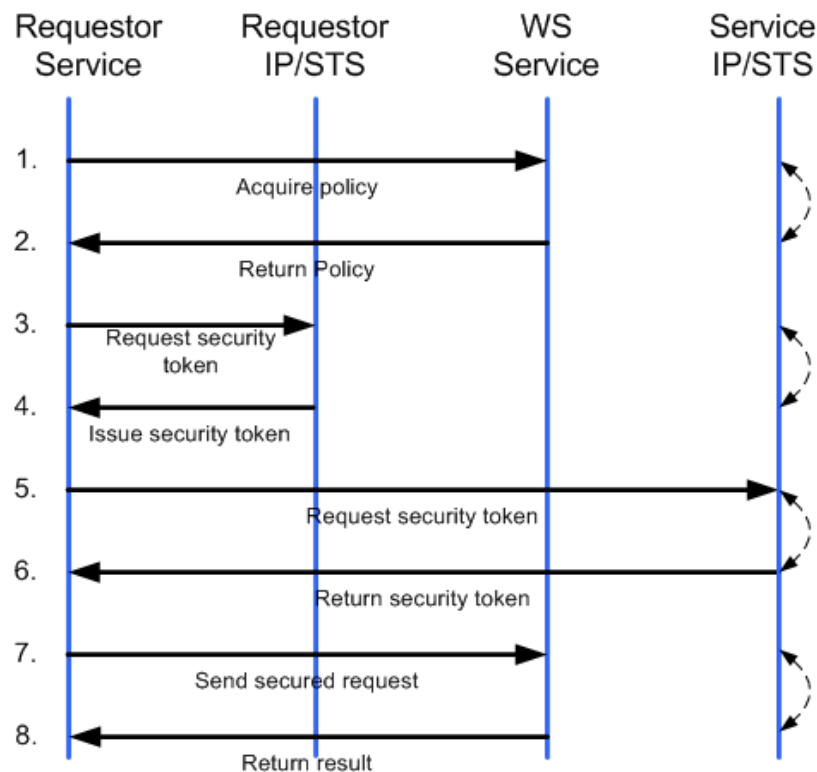
4174

4175

4176

4177

4178



4179

4180

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.

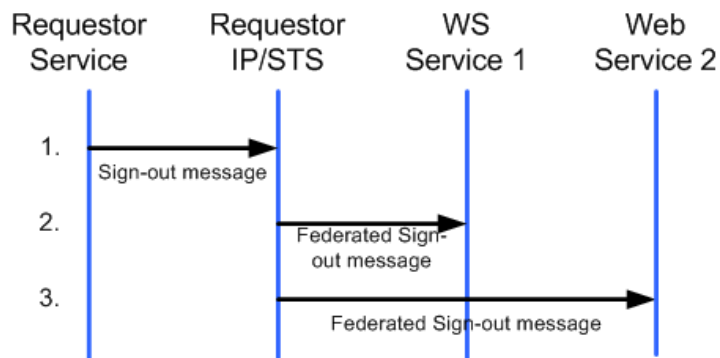
4181

4182 **C.2 Sign-Out**

4183 Just as it isn't typical for Web Service requestors to sign-in as a special operation, it isn't typical to *sign-*
4184 *out* either. However, for those scenarios where this is desirable, the sign-out messages defined in this
4185 specification can be used.

4186 In situations where federated sign-out messages are desirable, the requestor's IP/STS SHOULD keep
4187 track of the realms to which it has issued tokens – specifically the IP/STS for the realms (or resources if
4188 different). When the sign-out is received at the requestor's IP/STS, the requestor's IP/STS is responsible
4189 for issuing federated sign-out messages to interested and authorized parties. The exact mechanism by
4190 which this occurs is up to the IP/STS, but it is strongly RECOMMENDED that the sign-out messages
4191 defined in WS-Federation be used.

4192 When a federated sign-out message is received at a realm, the realm SHOULD clean-up any cached
4193 information and delete any associated state as illustrated in the figure below:

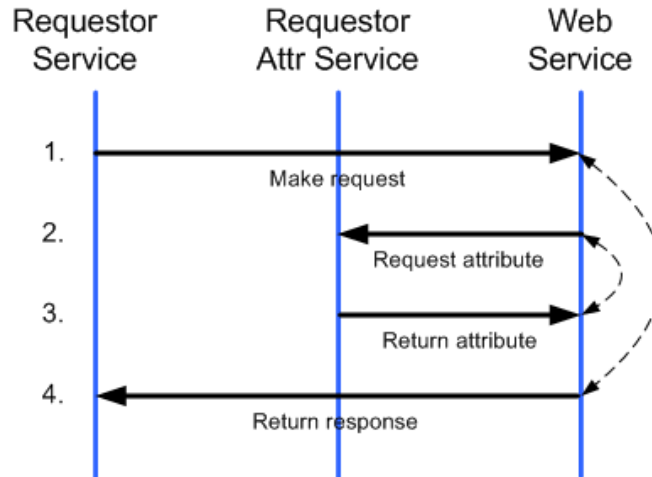


4194

4195 **C.3 Attributes**

4196 For Web Service requestors, attribute services are identified via WS-Policy or metadata as previously
4197 described. Web services and other authorized parties can obtain or even update attributes using the
4198 messages defined by the specific attribute service.

4199 The figure below illustrates a scenario where a requestor issues a request to a Web service. The request
4200 MAY include the requestor's policy or it may MAY be already cached at the service or the requestor MAY
4201 use [WS-MetadataExchange]. The Web service issues a request to the requestor's attribute service to
4202 obtain the values of a few attributes; WS-Policy MAY be used to describe the location of the attribute
4203 service. The service is authorized so the attributes are returned. The request is processed and a
4204 response is returned to the requestor.

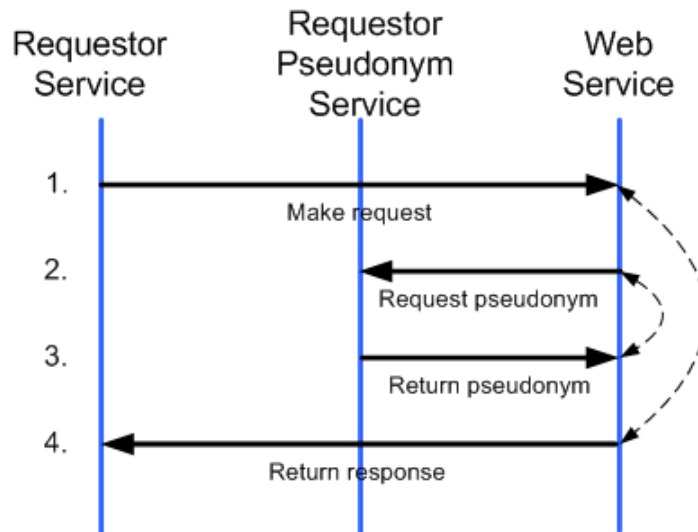


4205

4206 C.4 Pseudonyms

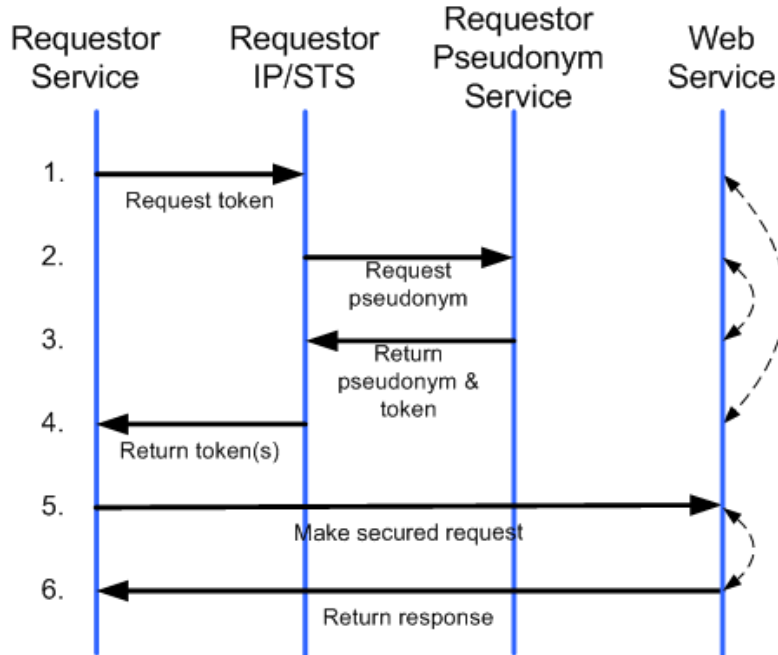
4207 For Web Service requestors, pseudonym services are identified via metadata as previously described.
 4208 Services and other authorized parties can obtain or manage pseudonyms using the messages previously
 4209 defined.

4210 The figure below illustrates a scenario where a requestor issues a request to a Web service. The request
 4211 MAY include the requestor's policy and the location of the requestor's pseudonym service or it MAY be
 4212 already cached at the Web service. The Web service issues a request to the requestor's pseudonyms
 4213 service to obtain the pseudonyms that are authorized by the security token. The Web service is
 4214 authorized so the pseudonym is returned. The request is processed and a response is returned to the
 4215 requestor.



4216

4217 As previously described, the pseudonym and IP/STS can interact as part of the token issuance process.
 4218 The figure below illustrates a scenario where a requestor has previously associated a pseudonym and a
 4219 security token for a specific realm. When the requestor requests a security token to the domain/realm,
 4220 the pseudonym and token are obtained and returned to the requestor. The requestor uses these security
 4221 tokens for accessing the Web service.

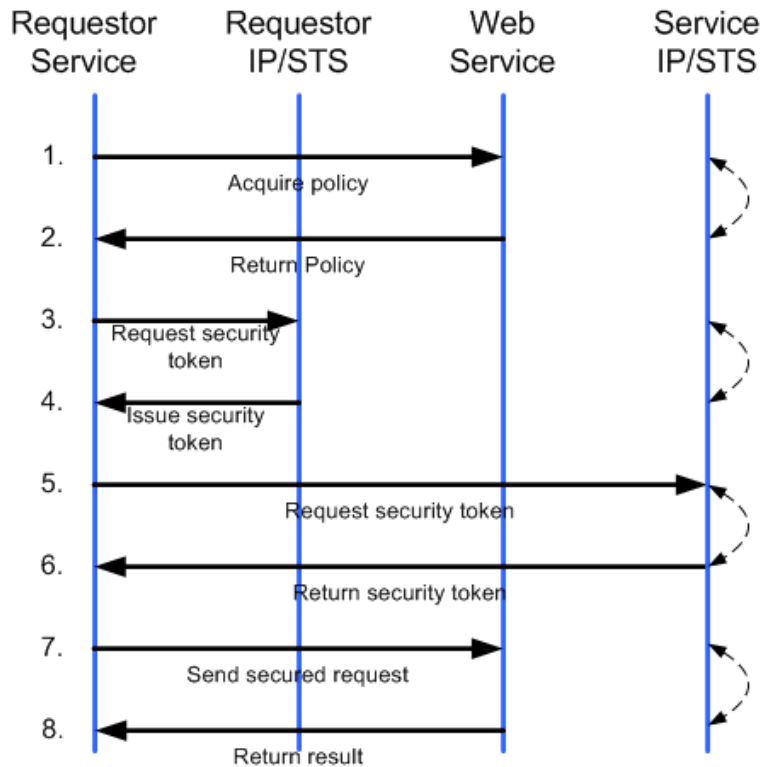


4222

4223 C.5 Detailed Example

4224 This section provides a detailed example of the protocol defined in this specification. The exact flow can
 4225 vary significantly; however, the following diagram and description depict a *common* sequence of events.

4226 In this scenario, a SOAP requestor is attempting to access a service which requires security
 4227 authentication to be validated by the resource's security token service.



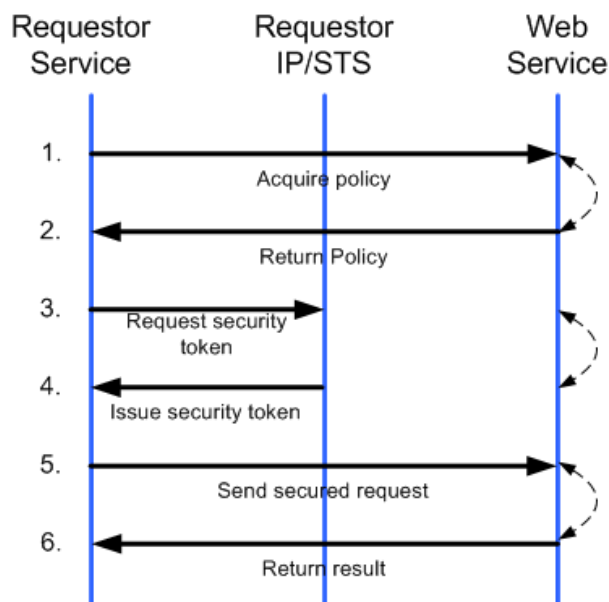
4228

- 4229 **Step 1: Acquire Policy**
- 4230 If the requestor doesn't already have the policy for the service, it can obtain it using the mechanisms
- 4231 defined in WS-MetadataExchange.
- 4232 **Step 2: Return Policy**
- 4233 The requested policy is returned using the mechanisms defined in WS-MetadataExchange.
- 4234 **Step 3: Request Security Token**
- 4235 The requestor requests a security token from its IP/STS (assuming short-lived security tokens) using the
- 4236 mechanisms defined in WS-Trust (<RequestSecurityToken>)
- 4237 **Step 4: Issue Security Token**
- 4238 The IP/STS returns a security token (and optional proof of possession information) using the mechanisms
- 4239 defined in WS-Trust (<RequestSecurityTokenResponse> and <RequestedProofToken>)
- 4240 **Step 5: Request Security Token**
- 4241 The requestor requests a security token from the Web services IP/STS for the target Web service using
- 4242 the mechanisms defined in WS-Trust (<RequestSecurityToken>). Note that this is determined via
- 4243 policy or some out-of-band mechanism.
- 4244 **Step 6: Issue Security Token**
- 4245 The Web service's IP/STS returns a token (and optionally proof of possession information) using the
- 4246 mechanisms defined in WS-Trust (<RequestSecurityTokenResponse>)
- 4247 **Step 7: Send secured request**
- 4248 The requestor sends the request to the service attaching and securing the message using the issued
- 4249 tokens as described in WS-Security.
- 4250 **Step 8: Return result**
- 4251 The service issues a secured reply using its security token.

4252 C.6 No Resource STS

4253 The figure below illustrates the resource access scenario above, but without a resource STS. That is, the

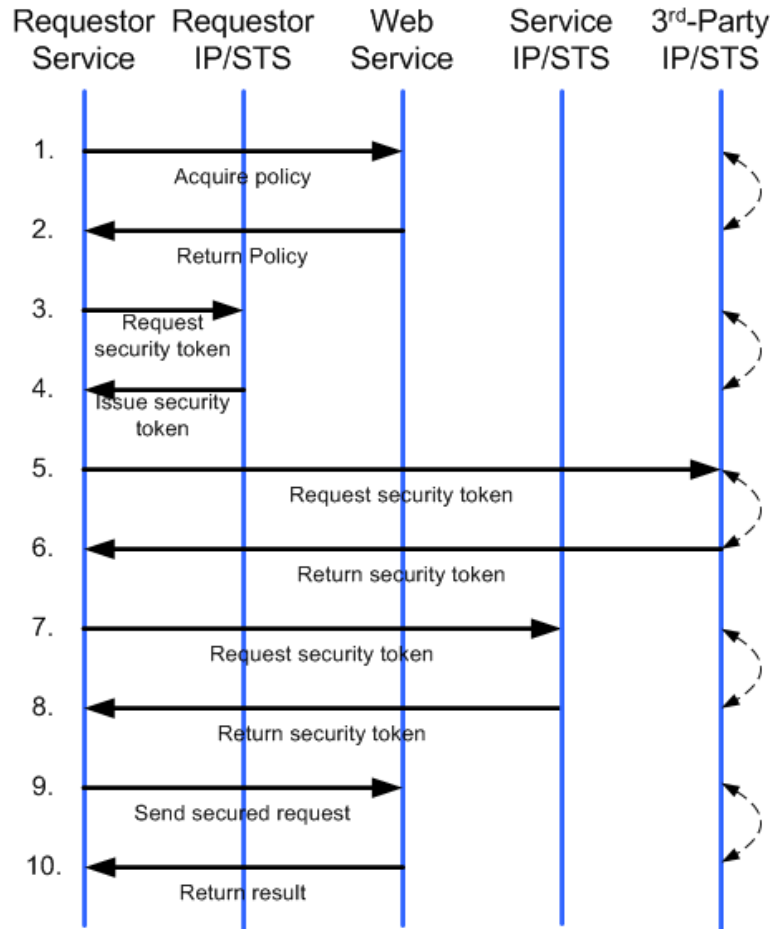
4254 Web service acts as its own STS:



4255

4256 **C.7 3rd-Party STS**

4257 The figure below illustrates the resource access scenario above, but trust is brokered through a 3rd-party
 4258 STS:

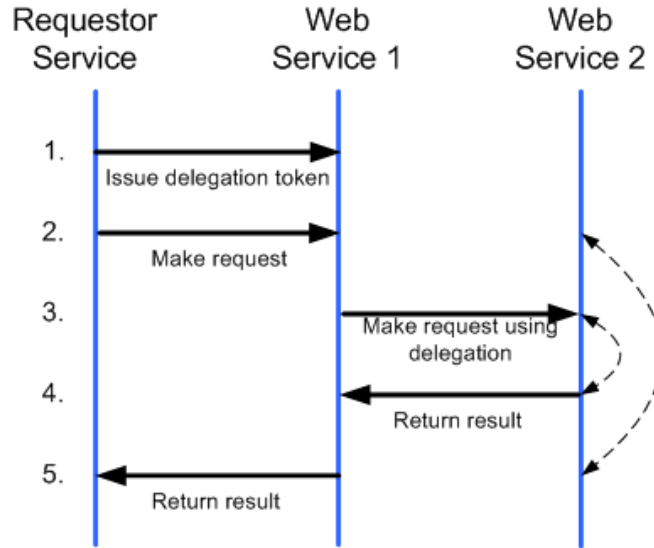


4259

4260 Note that 3rd-Party IP/STS is determined via policy or some out-of-band mechanism.

4261 **C.8 Delegated Resource Access**

4262 The figure below illustrates where a resource accesses data from another resource on behalf of the
 4263 requestor:



4264

4265 In this example, the requestor used a `<RequestSecurityTokenResponse>` as defined in WS-Trust to
 4266 issue the delegation token in Step 1. This provides to Web Service 1 the necessary information so that
 4267 Web Service 1 can act on the requestor's behalf as it contacts Web Service 2.

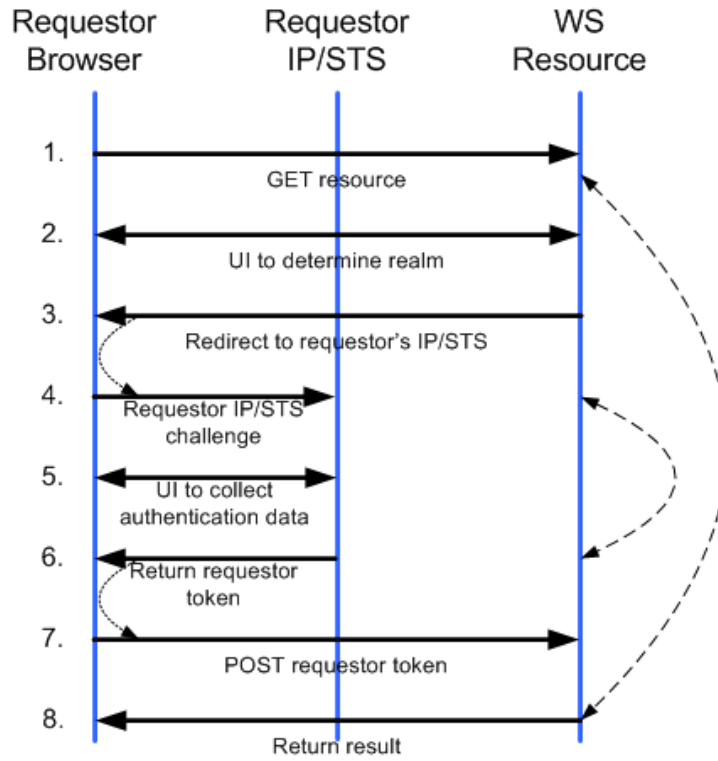
4268

4269 C.9 Additional Web Examples

4270 This section presents interaction diagrams for additional Web requestor scenarios.

4271 No Resource STS

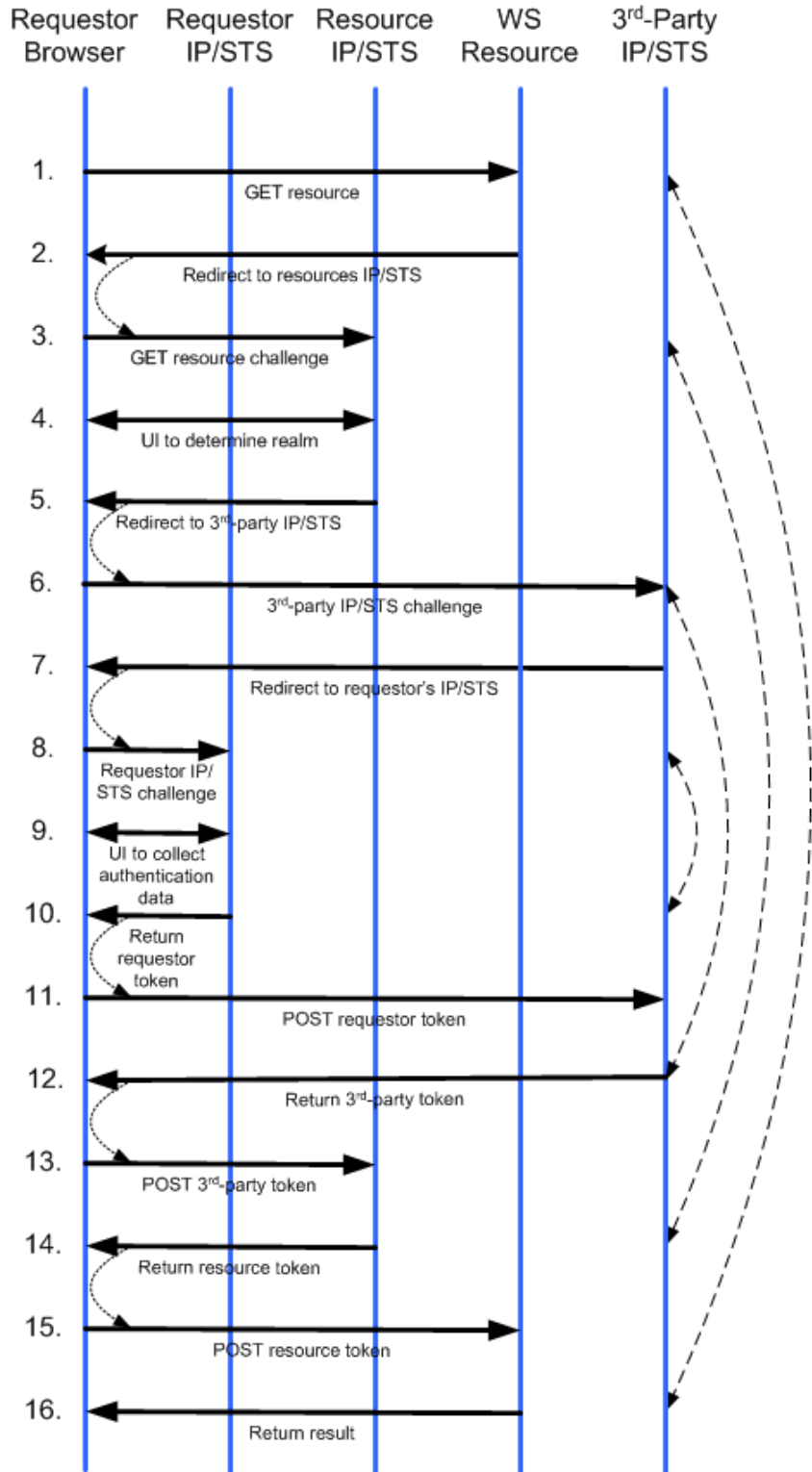
4272 The figure below illustrates the sign-in scenario above, but without a resource STS. That is, the requestor
 4273 acts as its own STS:



4274

4275 **3rd-Party STS**

4276 The figure below illustrates the sign-in scenario above, but trust is brokered through a 3rd-party STS:

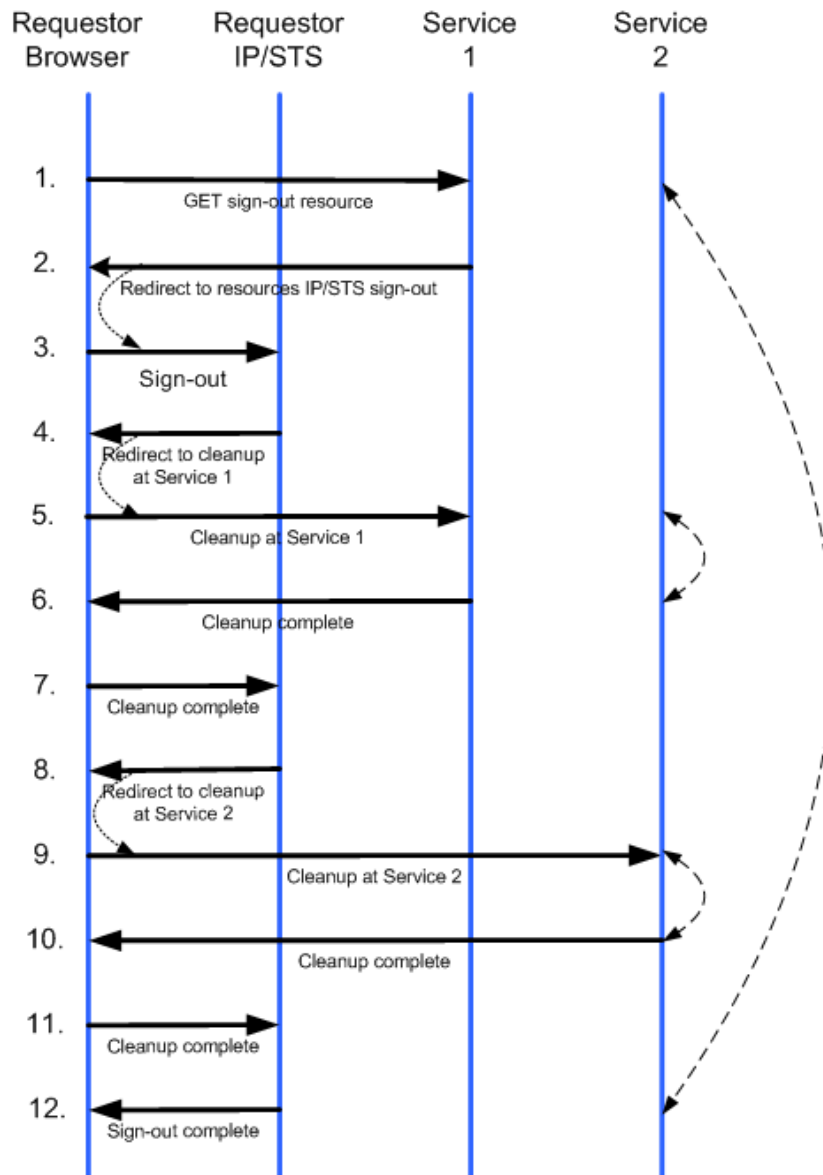


4277

4278 **Sign-Out**

4279 The figure below illustrates the sign-out flow for a Web browser requestor that has signed in at two sites
 4280 and requests that the sign-out cleanup requests redirect back to the requestor: The message flow is an

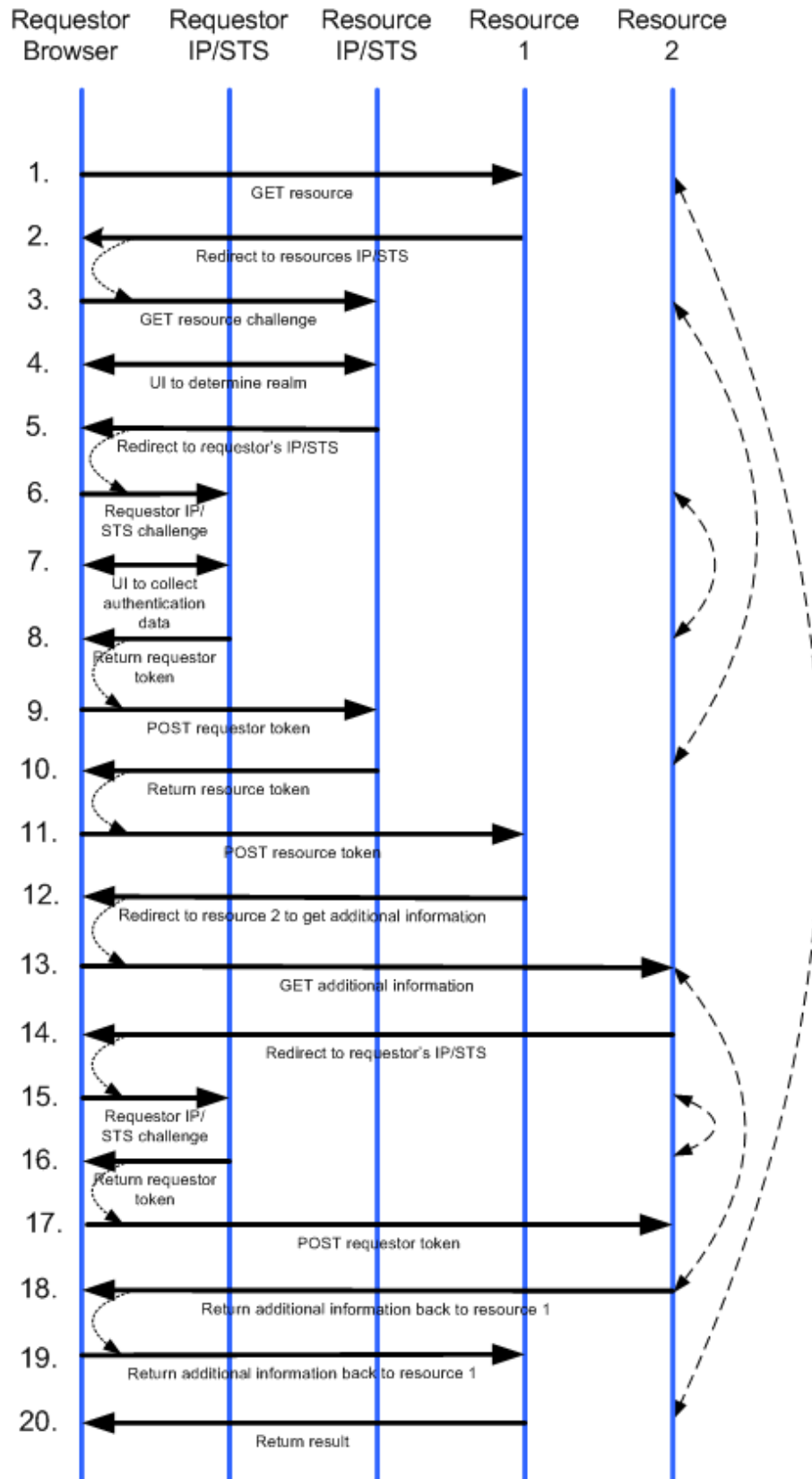
4281 example of the use case in which all sign-out messages must be transmitted by the requestor. Since it
 4282 cannot be assumed that all browser requestors can transmit parallel requests, the sequential method is
 4283 depicted. This message flow is enabled by the "wreply" parameter defined in section 13.2.4.



4284

4285 **Delegated Resource Access**

4286 The figure below illustrates the case where a resource accesses data from another resource on behalf of
 4287 the first resource and the information is returned through the requestor:



4289

Appendix D SAML Binding of Common Claims

4290

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.

4291

4292

4293

The display information, such as `auth:DisplayName`, `auth:Description` and `auth:DisplayValue` is not intended for serialization into tokens.

4294

4295

4296

Appendix E Acknowledgements

4297

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

4298

4299

Original Authors of the initial contributions:

4300

Hal Lockhart, BEA

4301

Steve Anderson, BMC Software

4302

Jeff Bohren, BMC Software

4303

Yakov Sverdlov, CA Inc.

4304

Maryann Hondo, IBM

4305

Hiroshi Maruyama, IBM

4306

Anthony Nadalin (Editor), IBM

4307

Nataraj Nagaratnam, IBM

4308

Toufic Boubez, Layer 7 Technologies, Inc.

4309

K Scott Morrison, Layer 7 Technologies, Inc.

4310

Chris Kaler (Editor), Microsoft

4311

Arun Nanda, Microsoft

4312

Don Schmidt, Microsoft

4313

Doug Walters, Microsoft

4314

Hervey Wilson, Microsoft

4315

Lloyd Burch, Novell, Inc.

4316

Doug Earl, Novell, Inc.

4317

Siddharth Bajaj, VeriSign

4318

Hemma Prafullchandra, VeriSign

4319

4320

Original Acknowledgements of the initial contributions:

4321

John Favazza, CA

4322

Tim Hahn, IBM

4323

Andrew Hatley, IBM

4324

Heather Hinton, IBM

4325

Michael McIntosh, IBM

4326

Anthony Moran, IBM

4327

Birgit Pfitzmann, IBM

4328

Bruce Rich, IBM

4329

Shane Weeden, IBM

4330

Jan Alexander, Microsoft

4331

Greg Carpenter, Microsoft

4332

Paul Cotton, Microsoft

4333

Marc Goodner, Microsoft

4334

Martin Gudgin, Microsoft

4335

Savas Parastatidis, Microsoft

4336

4337

TC Members during the development of this specification:

4338

Don Adams, TIBCO Software Inc.

4339

Steve Anderson, BMC Software

4340

Siddharth Bajaj, VeriSign

4341

Abbie Barbir, Nortel

4342

Hanane Becha, Nortel

4343

Toufic Boubez, Layer 7 Technologies Inc.

4344

Norman Brickman, Mitre Corporation

4345

Geoff Bullen, Microsoft Corporation

4346 Lloyd Burch, Novell
4347 Brian Campbell, Ping Identity Corporation
4348 Greg Carpenter, Microsoft Corporation
4349 Steve Carter, Novell
4350 Marco Carugi, Nortel
4351 Paul Cotton, Microsoft Corporation
4352 Doug Davis, IBM
4353 Fred Dushin, IONA Technologies
4354 Doug Earl, Novell
4355 Colleen Evans, Microsoft Corporation
4356 Christopher Ferris, IBM
4357 Marc Goodner, Microsoft Corporation
4358 Tony Gullotta, SOA Software Inc.
4359 Maryann Hondo, IBM
4360 Mike Kaiser, IBM
4361 Chris Kaler, Microsoft Corporation
4362 Paul Knight, Nortel
4363 Heather Kreger, IBM
4364 Ramanathan Krishnamurthy, IONA Technologies
4365 Kelvin Lawrence, IBM
4366 Paul Lesov, Wells Fargo
4367 David Lin, IBM
4368 Jonathan Marsh, WSO2
4369 Robin Martherus, Ping Identity Corporation
4370 Monica Martin, Microsoft Corporation
4371 Michael McIntosh, IBM
4372 Nandana Mihindukulasooriya, WSO2
4373 Anthony Nadalin, IBM
4374 Arun Nanda, Microsoft Corporation
4375 Kimberly Pease, Active Endpoints, Inc.
4376 Larry Rogers, Lockheed Martin
4377 Anil Saldhana, Red Hat
4378 Richard Sand, Tripod Technology Group, Inc.
4379 Don Schmidt, Microsoft Corporation
4380 Sidd Shenoy, Microsoft Corporation
4381 Kent Spaulding, Tripod Technology Group, Inc.
4382 David Staggs, Veterans Health Administration
4383 Yakov Sverdlov, CA
4384 Gene Thurston, AmberPoint
4385 Atul Tulshibagwale, Hewlett-Packard
4386 Ron Williams, IBM
4387 Jason Woloz, Booz Allen Hamilton
4388 Gerry Woods, SOA Software Inc.
4389