



WS-SecureConversation 1.3

Committee Draft 01, 06 September 2006

Artifact Identifier:

ws-secureconversation-1.3-spec-cd-01

Location:

Current: docs.oasis-open.org/ws-sx/ws-secureconversation/200512

This Version: docs.oasis-open.org/ws-sx/ws-secureconversation/200512

Previous Version: n/a

Artifact Type:

specification

Technical Committee:

OASIS Web Services Secure Exchange TC

Chair(s):

Kelvin Lawrence, IBM

Chris Kaler, Microsoft

Editor(s):

Anthony Nadalin, IBM

Marc Goodner, Microsoft

Martin Gudgin, Microsoft

Abbie Barbir, Nortel

Hans Granqvist, VeriSign

OASIS Conceptual Model topic area:

[Topic Area]

Related work:

NA

Abstract:

This specification defines extensions that build on [WS-Security] to provide a framework for requesting and issuing security tokens, and to broker trust relationships.

Status:

This document was last revised or approved by the WS-SX TC on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

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/ws-sx>.

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/ws-sx/ipr.php>).

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

Notices

Copyright © OASIS Open 2006. 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.

Table of Contents

1	Introduction	4
1.1	Goals and Non-Goals	4
1.2	Requirements	4
1.3	Namespace.....	4
1.4	Schema File	5
1.5	Terminology	5
1.5.1	Notational Conventions	6
1.6	Normative References	7
1.7	Non-Normative References	8
2	Security Context Token (SCT)	9
3	Establishing Security Contexts.....	12
3.1	SCT Binding of WS-Trust	13
3.2	SCT Request Example without Target Scope	13
3.3	SCT Request Example with Target Scope	14
3.4	SCT Propagation Example	16
4	Amending Contexts	17
5	Renewing Contexts	19
6	Canceling Contexts	21
7	Deriving Keys	23
7.1	Syntax	24
7.2	Examples	26
7.3	Implied Derived Keys	27
8	Associating a Security Context.....	29
9	Error Handling	31
10	Security Considerations	32
A.	Sample Usages	33
A.1	Anonymous SCT	33
A.2	Mutual Authentication SCT.....	34
B.	Token Discovery Using RST/RSTR	35
C.	Acknowledgements	36

1 Introduction

0 The mechanisms defined in [WS-Security] provide the basic mechanisms on top of which secure
1 messaging semantics can be defined for multiple message exchanges. This specification defines
2 extensions to allow security context establishment and sharing, and session key derivation. This allows
3 contexts to be established and potentially more efficient keys or new key material to be exchanged,
4 thereby increasing the overall performance and security of the subsequent exchanges.

5 The [WS-Security] specification focuses on the message authentication model. This approach, while
6 useful in many situations, is subject to several forms of attack (see Security Considerations section of
7 [WS-Security] specification).

8 Accordingly, this specification introduces a security context and its usage. The context authentication
9 model authenticates a series of messages thereby addressing these shortcomings, but requires
10 additional communications if authentication happens prior to normal application exchanges.

11
12 The security context is defined as a new [WS-Security] token type that is obtained using a binding of [WS-
13 Trust].

14
15 Compliant services are NOT REQUIRED to implement everything defined in this specification. However,
16 if a service implements an aspect of the specification, it MUST comply with the requirements specified
17 (e.g. related "MUST" statements).

1.1 Goals and Non-Goals

18
19 The primary goals of this specification are:

- 20 Define how security contexts are established
- 21 Describe how security contexts are amended
- 22 Specify how derived keys are computed and passed

23
24 It is not a goal of this specification to define how trust is established or determined.

25 This specification is intended to provide a flexible set of mechanisms that can be used to support a range
26 of security protocols. Some protocols may require separate mechanisms or restricted profiles of this
27 specification.

1.2 Requirements

28
29 The following list identifies the key driving requirements:

- 30 Derived keys and per-message keys
- 31 Extensible security contexts

1.3 Namespace

32
33 The [URI] that MUST be used by implementations of this specification is:

34 `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512`

35 Table 1 lists XML namespaces that are used in this specification. The choice of any namespace prefix is
36 arbitrary and not semantically significant.

37 Table 1: Prefixes and XML Namespaces used in this specification.

Prefix	Namespace	Specification(s)
S11	http://schemas.xmlsoap.org/soap/envelope/	[SOAP]
S12	http://www.w3.org/2003/05/soap-envelope	[SOAP12]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	[WS-Security]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	[WS-Security]
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512	[WS-Trust]
wsc	http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512	This specification
wsa	http://www.w3.org/2005/08/addressing	[WS-Addressing]
ds	http://www.w3.org/2000/09/xmldsig#	[XML-Signature]
xenc	http://www.w3.org/2001/04/xmlenc#	[XML-Encrypt]

38 1.4 Schema File

39 The schema [XML-Schema1], [XML-Schema2] for this specification can be located at:

40 [http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-](http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation.xsd)
 41 [secureconversation.xsd](http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation.xsd)

42
 43 In this document, reference is made to the `wsu:Id` attribute in the utility schema. These were added to
 44 the utility schema with the intent that other specifications requiring such an ID or timestamp could
 45 reference it (as is done here).

46 1.5 Terminology

47 **Claim** – A *claim* is a statement made about a client, service or other resource (e.g. name, identity, key,
 48 group, privilege, capability, etc.).

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

50 **Security Context** – A *security context* is an abstract concept that refers to an established authentication
 51 state and negotiated key(s) that may have additional security-related properties.

52 **Security Context Token** – A *security context token (SCT)* is a wire representation of that security context
 53 abstract concept, which allows a context to be named by a URI and used with [WS-Security].

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

56 **Proof-of-Possession Token** – A *proof-of-possession (POP) token* is a security token that contains
57 secret data that can be used to demonstrate authorized use of an associated security token. Typically,
58 although not exclusively, the proof-of-possession information is encrypted with a key known only to the
59 recipient of the POP token.

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

61 **Signature** - A *signature* [[XML-Signature](#)] is a value computed with a cryptographic algorithm and bound
62 to data in such a way that intended recipients of the data can use the signature to verify that the data has
63 not been altered and/or has originated from the signer of the message, providing message integrity and
64 authentication. The signature can be computed and verified with symmetric key algorithms, where the
65 same key is used for signing and verifying, or with asymmetric key algorithms, where different keys are
66 used for signing and verifying (a private and public key pair are used).

67 **Security Token Service** - A *security token service (STS)* is a Web service that issues security tokens
68 (see [[WS-Security](#)]). That is, it makes assertions based on evidence that it trusts, to whoever trusts it (or
69 to specific recipients). To communicate trust, a service requires proof, such as a signature, to prove
70 knowledge of a security token or set of security token. A service itself can generate tokens or it can rely
71 on a separate STS to issue a security token with its own trust statement (note that for some security token
72 formats this can just be a re-issuance or co-signature). This forms the basis of trust brokering.

73 **Request Security Token (RST)** – A *RST* is a message sent to a security token service to request a
74 security token.

75 **Request Security Token Response (RSTR)** – A *RSTR* is a response to a request for a security token.
76 In many cases this is a direct response from a security token service to a requestor after receiving an
77 RST message. However, in multi-exchange scenarios the requestor and security token service may
78 exchange multiple RSTR messages before the security token service issues a final RSTR message. One
79 or more RSTRs are contained within a single RequestSecurityTokenResponseCollection (RSTRC).

80 **1.5.1 Notational Conventions**

81 The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
82 NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described
83 in [[RFC2119](#)].

84

85 Namespace URIs of the general form "some-URI" represents some application-dependent or context-
86 dependent URI as defined in [[URI](#)].

87

88 This specification uses the following syntax to define outlines for messages:

89 The syntax appears as an XML instance, but values in italics indicate data types instead of literal
90 values.

91 Characters are appended to elements and attributes to indicate cardinality:

- 92 ○ "?" (0 or 1)
- 93 ○ "*" (0 or more)
- 94 ○ "+" (1 or more)

95 The character "|" is used to indicate a choice between alternatives.

96 The characters "(" and ")" are used to indicate that contained items are to be treated as a group
97 with respect to cardinality or choice.

98 The characters "[" and "]" are used to call out references and property names.

99 Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be
100 added at the indicated extension points but MUST NOT contradict the semantics of the parent

101 and/or owner, respectively. By default, if a receiver does not recognize an extension, the receiver
102 SHOULD ignore the extension; exceptions to this processing rule, if any, are clearly indicated
103 below.

104 XML namespace prefixes (see Table 1) are used to indicate the namespace of the element being
105 defined.

106

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

109 An element extensibility point is referred to using {any} in place of the element name. This
110 indicates that any element name can be used, from any namespace other than the namespace of
111 this specification.

112 An attribute extensibility point is referred to using @{any} in place of the attribute name. This
113 indicates that any attribute name can be used, from any namespace other than the namespace of
114 this specification.

115

116 In this document reference is made to the `wsu:Id` attribute and the `wsu:Created` and `wsu:Expires`
117 elements in a utility schema (<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd>). The `wsu:Id` attribute and the `wsu:Created` and `wsu:Expires` elements were added to the
118 utility schema with the intent that other specifications requiring such an ID type attribute or timestamp
119 element could reference it (as is done here).
120

121

122 1.6 Normative References

- 123 **[RFC2119]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC
124 2119, Harvard University, March 1997.
125 <http://www.ietf.org/rfc/rfc2119.txt> .
- 126 **[RFC2246]** IETF Standard, "The TLS Protocol", January 1999.
127 <http://www.ietf.org/rfc/rfc2246.txt>
- 128 **[SOAP]** W3C Note, "SOAP: Simple Object Access Protocol 1.1", 08 May 2000.
129 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
- 130 **[SOAP12]** W3C Recommendation, "SOAP 1.2 Part 1: Messaging Framework", 24 June
131 2003.
132 <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>
- 133 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI):
134 Generic Syntax", RFC 3986, MIT/LCS, Day Software, Adobe Systems, January
135 2005.
136 <http://www.ietf.org/rfc/rfc3986.txt>
- 137 **[WS-Addressing]** W3C Recommendation, "Web Services Addressing (WS-Addressing)", 9 May
138 2006.
139 <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509>.
- 140 **[WS-Security]** OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.0
141 (WS-Security 2004)", March 2004.
142 <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- 143
144 OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.1
145 (WS-Security 2004)", February 2006.
146 <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- 147
148 **[WS-Trust]** OASIS Committee Draft, "WS-Trust 1.3", September 2006
149 <http://docs.oasis-open.org/ws-sx/ws-trust/200512>

150 **[XML-Encrypt]** W3C Recommendation, "XML Encryption Syntax and Processing", 10
151 December 2002.
152 <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>.
153 **[XML-Schema1]** W3C Recommendation, "XML Schema Part 1: Structures Second Edition", 28
154 October 2004.
155 <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>.
156 **[XML-Schema2]** W3C Recommendation, "XML Schema Part 2: Datatypes Second Edition", 28
157 October 2004.
158 <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.
159 **[XML-Signature]** W3C Recommendation, "XML-Signature Syntax and Processing", 12 February
160 2002.
161 <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>

162 **1.7 Non-Normative References**

163 **[WS-MEX]** "Web Services Metadata Exchange (WS-MetadataExchange)", BEA, Computer
164 Associates, IBM, Microsoft, SAP, Sun Microsystems, Inc., webMethods,
165 September 2004.
166 **[WS-Policy]** W3C Member Submission, "Web Services Policy 1.2 - Framework", 25 April
167 2006.
168 <http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/>
169 **[WS-PolicyAttachment]** W3C Member Submission, "Web Services Policy 1.2 - Attachment", 25 April
170 2006.
171 <http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/>

2 Security Context Token (SCT)

While message authentication is useful for simple or one-way messages, parties that wish to exchange multiple messages typically establish a security context in which to exchange multiple messages. A security context is shared among the communicating parties for the lifetime of a communications session.

In this specification, a security context is represented by the `<wsc:SecurityContextToken>` security token. In the [WS-Security] and [WS-Trust] framework, the following URI is used to represent the token type:

```
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
```

The Security Context Token does not support references to it using key identifiers or key names. All references MUST either use an ID (to a `wsu:Id` attribute) or a `<wsse:Reference>` to the `<wsc:Identifier>` element.

Once the context and secret have been established (authenticated), the mechanisms described in [Derived Keys](#) can be used to compute derived keys for each key usage in the secure context.

The following illustration represents an overview of the syntax of the `<wsc:SecurityContextToken>` element. It should be noted that this token supports an open content model to allow context-specific data to be passed.

```
<wsc:SecurityContextToken wsu:Id="..." xmlns:wsc="..." xmlns:wsu="..." ...>
  <wsc:Identifier>...</wsc:Identifier>
  <wsc:Instance>...</wsc:Instance>
  ...
</wsc:SecurityContextToken>
```

The following describes elements and attributes used in a `<wsc:SecurityContextToken>` element.

`/wsc:SecurityContextToken`

This element is a security token that describes a security context.

`/wsc:SecurityContextToken/wsc:Identifier`

This required element identifies the security context using an absolute URI. Each security context URI MUST be unique to both the sender and recipient. It is RECOMMENDED that the value be globally unique in time and space.

`/wsc:SecurityContextToken/wsc:Instance`

When contexts are renewed and given different keys it is necessary to identify the different key instances without revealing the actual key. When present this optional element contains a string that is unique for a given key value for this `wsc:Identifier`. The initial issuance need not contain a `wsc:Instance` element, however, all subsequent issuances with different keys MUST have a `wsc:Instance` element with a unique value.

`/wsc:SecurityContextToken/@wsu:Id`

This optional attribute specifies a string label for this element.

`/wsc:SecurityContextToken/@{any}`

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

216 /wsc:SecurityContextToken/{any}

217 This is an extensibility mechanism to allow additional elements (arbitrary content) to be used.

218

219 The <wsc:SecurityContextToken> token elements MUST be preserved. That is, whatever elements
220 contained within the tag on creation MUST be preserved wherever the token is used. A consumer of a
221 <wsc:SecurityContextToken> token MAY extend the token by appending information.
222 Consequently producers of <wsc:SecurityContextToken> tokens should consider this fact when
223 processing previously generated tokens. A service consuming (processing) a
224 <wsc:SecurityContextToken> token MAY fault if it discovers an element or attribute inside the token
225 that it doesn't understand, or it MAY ignore it. The fault code wsc:UnsupportedContextToken is
226 RECOMMENDED if a fault is raised. The behavior is specified by the services policy [WS-Policy] [WS-
227 PolicyAttachment]. Care should be taken when adding information to tokens to ensure that relying parties
228 can ensure the information has not been altered since the SCT definition does not require a specific way
229 to secure its contents (which as noted above can be appended to).

230

231 Security contexts, like all security tokens, can be referenced using the mechanisms described in [WS-
232 Security] (the <wsse:SecurityTokenReference> element referencing the wsu:Id attribute relative to
233 the XML base document or referencing using the <wsc:Identifier> element's absolute URI). When a
234 token is referenced, the associated key is used. If a token provides multiple keys then specific bindings
235 and profiles must describe how to reference the separate keys. If a specific key instance needs to be
236 referenced, then the global attribute wsc:Instance is included in the <wsse:Reference> sub-element
237 (only when using <wsc:Identifier> references) of the <wsse:SecurityTokenReference>
238 element as illustrated below:

```
239 <wsse:SecurityTokenReference xmlns:wsse="..." xmlns:wsc="...">  
240 <wsse:Reference URI="uuid:... " wsc:Instance="..."/>  
241 </wsse:SecurityTokenReference>
```

242

243 The following sample message illustrates the use of a security context token. In this example a context
244 has been established and the secret is known to both parties. This secret is used to sign the message
245 body.

```
246 (001) <?xml version="1.0" encoding="utf-8"?>  
247 (002) <S11:Envelope xmlns:S11="..." xmlns:ds="..." xmlns:wsse="..."  
248 < xmlns:wsu="..." xmlns:wsc="...">  
249 (003) <S11:Header>  
250 (004) ...  
251 (005) <wsse:Security>  
252 (006) <wsc:SecurityContextToken wsu:Id="MyID">  
253 (007) <wsc:Identifier>uuid:...</wsc:Identifier>  
254 (008) </wsc:SecurityContextToken>  
255 (009) <ds:Signature>  
256 (010) ...  
257 (011) <ds:KeyInfo>  
258 (012) <wsse:SecurityTokenReference>  
259 (013) <wsse:Reference URI="#MyID"/>  
260 (014) </wsse:SecurityTokenReference>  
261 (015) </ds:KeyInfo>  
262 (016) </ds:Signature>  
263 (017) </wsse:Security>  
264 (018) </S11:Header>  
265 (019) <S11:Body wsu:Id="MsgBody">
```

266
267
268
269
270
271

```
(020)      <tru:StockSymbol
           xmlns:tru="http://fabrikam123.com/payloads">
           QQQ
           </tru:StockSymbol>
(021)      </S11:Body>
(022) </S11:Envelope>
```

272

273 Let's review some of the key sections of this example:

274 Lines (003)-(018) contain the SOAP message headers.

275 Lines (005)-(017) represent the `<wsse:Security>` header block. This contains the security-related information for the message.

277 Lines (006)-(008) specify a [security token](#) that is associated with the message. In this case it is a security context token. Line (007) specifies the unique ID of the context.

279 Lines (009)-(016) specify the digital signature. In this example, the signature is based on the security context (specifically the secret/key associated with the context). Line (010) represents the typical contents of an XML Digital Signature which, in this case, references the body and potentially some of the other headers expressed by line (004).

283

284 Lines (012)-(014) indicate the key that was used for the signature. In this case, it is the security context token included in the message. Line (013) provides a URI link to the security context token specified in Lines (006)-(008).

287 The body of the message is represented by lines (019)-(021).

288

3 Establishing Security Contexts

289 A security context needs to be created and shared by the communicating parties before being used. This
290 specification defines three different ways of establishing a security context among the parties of a secure
291 communication.

292

293 **Security context token created by a security token service** – The context initiator asks a security
294 token service to create a new security context token. The newly created security context token is
295 distributed to the parties through the mechanisms defined here and in [WS-Trust]. For this scenario the
296 initiating party sends a `<wst:RequestSecurityToken>` request to the token service and a
297 `<wst:RequestSecurityTokenResponseCollection>` containing a
298 `<wst:RequestSecurityTokenResponse>` is returned. The response contains a
299 `<wst:RequestedSecurityToken>` containing (or pointing to) the new security context token and a
300 `<wst:RequestedProofToken>` pointing to the "secret" for the returned context. The requestor then
301 uses the security context token (with [WS-Security]) when securing messages to applicable services.

302

303 **Security context token created by one of the communicating parties and propagated with a**
304 **message** – The initiator creates a security context token and sends it to the other parties on a message
305 using the mechanisms described in this specification and in [WS-Trust]. This model works when the
306 sender is trusted to always create a new security context token. For this scenario the initiating party
307 creates a security context token and issues a signed unsolicited
308 `<wst:RequestSecurityTokenResponse>` to the other party. The message contains a
309 `<wst:RequestedSecurityToken>` containing (or pointing to) the new security context token and a
310 `<wst:RequestedProofToken>` pointing to the "secret" for the security context token. The recipient
311 can then choose whether or not to accept the security context token. As described in [WS-Trust], the
312 `<wst:RequestSecurityTokenResponse>` element MAY be in the
313 `<wst:RequestSecurityTokenResponseCollection>` within a body or inside a header block. It
314 should be noted that unless delegation tokens are used, this scenario requires that parties trust each
315 other to share a secret key (and non-repudiation is probably not possible). As receipt of these messages
316 may be expensive, and because a recipient may receive multiple messages, the
317 ... /wst:RequestSecurityTokenResponse/@Context attribute in [WS-Trust] allows the initiator to specify a
318 URI to indicate the intended usage (allowing processing to be optimized).

319

320 **Security context token created through negotiation/exchanges** – When there is a need to negotiate
321 or participate in a sequence of message exchanges among the participants on the contents of the
322 security context token, such as the shared secret, this specification allows the parties to exchange data to
323 establish a security context. For this scenario the initiating party sends a
324 `<wst:RequestSecurityToken>` request to the other party and a
325 `<wst:RequestSecurityTokenResponse>` is returned. It is RECOMMENDED that the framework
326 described in [WS-Trust] be used; however, the type of exchange will likely vary. If appropriate, the basic
327 challenge-response definition in [WS-Trust] is RECOMMENDED. Ultimately (if successful), a final
328 response contains a `<wst:RequestedSecurityToken>` containing (or pointing to) the new security
329 context and a `<wst:RequestedProofToken>` pointing to the "secret" for the context.

330 If an SCT is received, but the key sizes are not supported, then a fault SHOULD be generated using the
331 `wsc:UnsupportedContextToken` fault code unless another more specific fault code is available.

3.1 SCT Binding of WS-Trust

This binding describes how to use [WS-Trust] to request and return SCTs. This binding builds on the issuance binding for [WS-Trust] (note that other sections of this specification define new separate bindings of [WS-Trust]). Consequently, aspects of the issuance binding apply to this binding unless otherwise stated. For example, the token request type is the same as in the issuance binding.

When requesting and returning security context tokens the following Action URIs [WS-Addressing] are used (note that a specialized action is used here because of the specialized semantics of SCTs):

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT
```

As with all token services, the options supported may be limited. This is especially true of SCTs because the issuer may only be able to issue tokens for itself and quite often will only support a specific set of algorithms and parameters as expressed in its policy.

SCTs are not required to have lifetime semantics. That is, some SCTs may have specific lifetimes and others may be bound to other resources rather than have their own lifetimes.

Since the SCT binding builds on the issuance binding, it allows the optional extensions defined for the issuance binding including the use of exchanges. Subsequent profiles MAY restrict the extensions and types and usage of exchanges.

3.2 SCT Request Example without Target Scope

The following illustrates a request for a SCT from a security token service. The request in this example contains no information concerning the Web Service with whom the requestor wants to communicate securely (e.g. using the `wsp:AppliesTo` parameter in the RST). In order for the security token service to process this request it must have prior knowledge for which Web Service the requestor needs a token. This may be preconfigured although it is typically passed in the RST. In this example the key is encrypted for the recipient (security token service) using the token service's X.509 certificate as per XML Encryption [XML-Encrypt]. The encrypted data (using the encrypted key) contains a `<wsse:UsernameToken>` token that the recipient uses to authorize the request. The request is secured (integrity) using the X.509 certificate of the requestor. The response encrypts the proof information using the requestor's X.509 certificate and secures the message (integrity) using the token service's X.509 certificate. Note that the details of XML Signature and XML Encryption have been omitted; refer to [WS-Security] for additional details. It should be noted that if the requestor doesn't have an X.509 this scenario could be achieved using a TLS [RFC2246] connection or by creating an ephemeral key.

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
  xmlns:wst="..." xmlns:xenc="...">
  <S11:Header>
    ...
    <wsa:Action xmlns:wsa="...">
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT
    </wsa:Action>
    ...
    <wsse:Security>
      <xenc:EncryptedKey>
        ...
      </xenc:EncryptedKey>
      <xenc:EncryptedData Id="encUsernameToken">
        .. encrypted username token (whose id is myToken) ..
      </xenc:EncryptedData>
      <ds:Signature xmlns:ds="...">
```

```

381     ...
382     <ds:KeyInfo>
383         <wsse:SecurityTokenReference>
384             <wsse:Reference URI="#myToken"/>
385         </wsse:SecurityTokenReference>
386     </ds:KeyInfo>
387     </ds:Signature>
388 </wsse:Security>
389     ...
390 </S11:Header>
391 <S11:Body wsu:Id="req">
392     <wst:RequestSecurityToken>
393         <wst:TokenType>
394             http://docs.oasis-open.org/ws-sx/ws-
395 secureconversation/200512/sct
396         </wst:TokenType>
397         <wst:RequestType>
398             http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
399         </wst:RequestType>
400     </wst:RequestSecurityToken>
401 </S11:Body>
402 </S11:Envelope>

```

403

```

404 <S11:Envelope xmlns:S11="..."
405     xmlns:wst="..." xmlns:wsc="..." xmlns:xenc="...">
406     <S11:Header>
407         ...
408         <wsa:Action xmlns:wsa="...">
409             http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT
410         </wsa:Action>
411         ...
412     </S11:Header>
413     <S11:Body>
414         <wst:RequestSecurityTokenResponseCollection>
415             <wst:RequestSecurityTokenResponse>
416                 <wst:RequestedSecurityToken>
417                     <wsc:SecurityContextToken>
418                         <wsc:Identifier>uuid:...</wsc:Identifier>
419                     </wsc:SecurityContextToken>
420                 </wst:RequestedSecurityToken>
421                 <wst:RequestedProofToken>
422                     <xenc:EncryptedKey Id="newProof">
423                         ...
424                     </xenc:EncryptedKey>
425                 </wst:RequestedProofToken>
426             </wst:RequestSecurityTokenResponse>
427         </wst:RequestSecurityTokenResponseCollection>
428     </S11:Body>
429 </S11:Envelope>

```

430 3.3 SCT Request Example with Target Scope

431 There are scenarios where a security token service is used to broker trust using SCT tokens between
432 requestors and Web Services endpoints. In these cases it is typical for requestors to identify the target
433 Web Service in the RST.

434 In the example below the requestor uses the element <wsp:AppliesTo> with an endpoint reference as
435 described in [WS-Trust] in the SCT request to indicate the Web Service the token is needed for.

436 In the request example below the <wst:TokenType> element is omitted. This requires that the security
437 token service know what type of token the endpoint referenced in the <wsp:AppliesTo> element expects.

```

438 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
439     xmlns:wst="..." xmlns:xenc="..." xmlns:wsp="..." xmlns:wsa="...">
440   <S11:Header>
441     ...
442     <wsa:Action xmlns:wsa="...">
443       http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT
444     </wsa:Action>
445     ...
446     <wsse:Security>
447       ...
448     </wsse:Security>
449     ...
450   </S11:Header>
451   <S11:Body wsu:Id="req">
452     <wst:RequestSecurityToken>
453       <wst:RequestType>
454         http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
455       </wst:RequestType>
456       <wsp:AppliesTo>
457         <wsa:EndpointReference>
458           <wsa:Address>http://example.org/webservice</wsa:Address>
459         </wsa:EndpointReference>
460       </wsp:AppliesTo>
461     </wst:RequestSecurityToken>
462   </S11:Body>
463 </S11:Envelope>

```

464

```

465 <S11:Envelope xmlns:S11="..."
466     xmlns:wst="..." xmlns:wsc="..." xmlns:xenc="..." xmlns:wsp="..."
467     xmlns:wsa="...">
468   <S11:Header>
469     <wsa:Action xmlns:wsa="...">
470       http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT
471     </wsa:Action>
472     ...
473   </S11:Header>
474   <S11:Body>
475     <wst:RequestSecurityTokenResponseCollection>
476       <wst:RequestSecurityTokenResponse>
477         <wst:RequestedSecurityToken>
478           <wsc:SecurityContextToken>
479             <wsc:Identifier>uuid:...</wsc:Identifier>
480           </wsc:SecurityContextToken>
481         </wst:RequestedSecurityToken>
482         <wst:RequestedProofToken>
483           <xenc:EncryptedKey Id="newProof">
484             ...
485           </xenc:EncryptedKey>
486         </wst:RequestedProofToken>
487         <wsp:AppliesTo>
488           <wsa:EndpointReference>
489             <wsa:Address>http://example.org/webservice</wsa:Address>
490           </wsa:EndpointReference>
491         </wsp:AppliesTo>
492       </wst:RequestSecurityTokenResponse>
493     </wst:RequestSecurityTokenResponseCollection>
494   </S11:Body>
495 </S11:Envelope>

```

496

497 3.4 SCT Propagation Example

498 The following illustrates propagating a context to another party. This example does not contain any
499 information regarding the Web Service the SCT is intended for (e.g. using the `wsp:AppliesTo` parameter
500 in the RST).

```
501 <S11:Envelope xmlns:S11="..."  
502     xmlns:wst="..." xmlns:wsc="..." xmlns:xenc="..." >  
503   <S11:Header>  
504     ...  
505   </S11:Header>  
506   <S11:Body>  
507     <wst:RequestSecurityTokenResponse>  
508       <wst:RequestedSecurityToken>  
509         <wsc:SecurityContextToken>  
510           <wsc:Identifier>uuid:...</wsc:Identifier>  
511         </wsc:SecurityContextToken>  
512       </wst:RequestedSecurityToken>  
513       <wst:RequestedProofToken>  
514         <xenc:EncryptedKey Id="newProof">  
515           ...  
516         </xenc:EncryptedKey>  
517       </wst:RequestedProofToken>  
518     </wst:RequestSecurityTokenResponse>  
519   </S11:Body>  
520 </S11:Envelope>
```

521

4 Amending Contexts

522 When an SCT is created, a set of claims is associated with it. There are times when an existing SCT
523 needs to be amended to carry additional claims (note that the decision as to who is authorized to amend
524 a context is a service-specific decision). This is done using the SCT Amend binding. In such cases an
525 explicit request is made to amend the claims associated with an SCT. It should be noted that using the
526 mechanisms described in [WS-Trust], an issuer MAY, at any time, return an amended SCT by issuing an
527 unsolicited (not explicitly requested) SCT inside an RSTR (either as a separate message or in a header).

528 The following Action URIs are used with this binding:

529
530

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Amend  
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Amend
```

531

532 This binding allows optional extensions but DOES NOT allow key semantics to be altered.

533 Proof of possession of the key associated with the security context MUST be proven in order for context
534 to be amended. It is RECOMMENDED that the proof of possession is done by creating a signature over
535 the message body and key headers using the key associated with the security context.

536 Additional claims to amend the security context with MUST be indicated by providing signatures over the
537 security context signature created using the key associated with the security context. Those additional
538 signatures are used to prove additional security tokens that carry claims to augment the security context.

539 This binding uses the request type from the issuance binding.

540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
  xmlns:wst="..." xmlns:wsc="...">  
  <S11:Header>  
    ...  
    <wsa:Action xmlns:wsa="...">  
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Amend  
    </wsa:Action>  
    ...  
    <wsse:Security>  
      <xx:CustomToken wsu:Id="cust" xmlns:xx="...">  
        ...  
      </xx:CustomToken>  
      <ds:Signature xmlns:ds="...">  
        ...signature over #sig1 using #cust...  
      </ds:Signature>  
      <wsc:SecurityContextToken wsu:Id="sct">  
        <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>  
      </wsc:SecurityContextToken>  
      <ds:Signature xmlns:ds="..." Id="sig1">  
        ...signature over body and key headers using #sct...  
      <ds:KeyInfo>  
        <wsse:SecurityTokenReference>  
          <wsse:Reference URI="#sct"/>  
        </wsse:SecurityTokenReference>  
      </ds:KeyInfo>  
      ...  
    </ds:Signature>  
  </wsse:Security>  
  ...  
</S11:Header>
```

```
570 <S11:Body wsu:Id="req">
571   <wst:RequestSecurityToken>
572     <wst:RequestType>
573       http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
574     </wst:RequestType>
575   </wst:RequestSecurityToken>
576 </S11:Body>
577 </S11:Envelope>
```

```
578
579 <S11:Envelope xmlns:S11="..." xmlns:wst="..." xmlns:wsc="...">
580   <S11:Header>
581     ...
582     <wsa:Action xmlns:wsa="...">
583       http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Amend
584     </wsa:Action>
585     ...
586   </S11:Header>
587   <S11:Body>
588     <wst:RequestSecurityTokenResponseCollection>
589       <wst:RequestSecurityTokenResponse>
590         <wst:RequestedSecurityToken>
591           <wsc:SecurityContextToken>
592             <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
593           </wsc:SecurityContextToken>
594         </wst:RequestedSecurityToken>
595       </wst:RequestSecurityTokenResponse>
596     </wst:RequestSecurityTokenResponseCollection>
597   </S11:Body>
598 </S11:Envelope>
```

599

5 Renewing Contexts

600 When a security context is created it typically has an associated expiration. If a requestor desires to
601 extend the duration of the token it uses a custom binding of the renewal mechanism defined in WS-Trust.
602 The following Action URIs are used with this binding:

603
604

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Renew
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Renew
```

605

606 This binding allows optional extensions but DOES NOT allow key semantics to be altered.

607 A renewal MUST include re-authentication of the original claims because the original claims might have
608 an expiration time that conflicts with the requested expiration time in the renewal request. Because the
609 security context token issuer is not required to cache such information from the original issuance request,
610 the requestor is required to re-authenticate the original claims in every renewal request. It is
611 RECOMMENDED that the original claims re-authentication is done in the same way as in the original
612 token issuance request.

613 Proof of possession of the key associated with the security context MUST be proven in order for security
614 context to be renewed. It is RECOMMENDED that this is done by creating the original claims signature
615 over the signature that signs message body and key headers.

616 During renewal, new key material MAY be exchanged. Such key material MUST NOT be protected using
617 the existing session key.

618 This binding uses the request type from the renewal binding.

619 The following example illustrates a renewal which re-proves the original claims.

620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
  xmlns:wst="..." xmlns:wsc="...">
  <S11:Header>
    ...
    <wsa:Action xmlns:wsa="...">
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Renew
    </wsa:Action>
    ...
    <wsse:Security>
      <xx:CustomToken wsu:Id="cust" xmlns:xx="...">
        ...
      </xx:CustomToken>
      <ds:Signature xmlns:ds="..." Id="sig1">
        ... signature over body and key headers using #cust...
      </ds:Signature>
      <wsc:SecurityContextToken wsu:Id="sct">
        <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
      </wsc:SecurityContextToken>
      <ds:Signature xmlns:ds="..." Id="sig2">
        ... signature over #sig1 using #sct ...
      </ds:Signature>
    </wsse:Security>
    ...
  </S11:Header>
  <S11:Body wsu:Id="req">
    <wst:RequestSecurityToken>
      <wst:RequestType>
```

```
647         http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew
648     </wst:RequestType>
649     <wst:RenewTarget>
650         <wsse:SecurityTokenReference>
651             <wsse:Reference URI="#sct"/>
652         </wsse:SecurityTokenReference>
653     </wst:RenewTarget>
654     <wst:Lifetime>...</wst:Lifetime>
655 </wst:RequestSecurityToken>
656 </S11:Body>
657 </S11:Envelope>
```

658

```
659 <S11:Envelope xmlns:S11="..." xmlns:wst="..." xmlns:wsc="...">
660     <S11:Header>
661         ...
662         <wsa:Action xmlns:wsa="...">
663             http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Renew
664         </wsa:Action>
665         ...
666     </S11:Header>
667     <S11:Body>
668         <wst:RequestSecurityTokenResponseCollection>
669             <wst:RequestSecurityTokenResponse>
670                 <wst:RequestedSecurityToken>
671                     <wsc:SecurityContextToken>
672                         <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
673                         <wsc:Instance>UUID2</wsc:Instance>
674                     </wsc:SecurityContextToken>
675                 </wst:RequestedSecurityToken>
676                 <wst:Lifetime>...</wst:Lifetime>
677             </wst:RequestSecurityTokenResponse>
678         </wst:RequestSecurityTokenResponseCollection>
679     </S11:Body>
680 </S11:Envelope>
```

6 Canceling Contexts

681

682 It is not uncommon for a requestor to be done with a security context token before it expires. In such
683 cases the requestor can explicitly cancel the security context using this specialized binding based on the
684 WS-Trust Cancel binding.

685 The following Action URIs are used with this binding:

686
687

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Cancel  
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Cancel
```

688

689 Once a security context has been cancelled it MUST NOT be allowed for authentication or authorization
690 or allow renewal.

691

692 Proof of possession of the key associated with the security context MUST be proven in order for security
693 context to be cancelled. It is RECOMMENDED that this is done by creating a signature over the message
694 body and key headers using the key associated with the security context.

695

696 This binding uses the Cancel request type from WS-Trust.

697

698 As described in WS-Trust the RSTR cancel message is informational and the context is cancelled once
699 the cancel RST is processed even in the cancel RSTR is never received by the requestor.

700

701 The following example illustrates canceling a context.

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
  xmlns:wst="..." xmlns:wsc="...">  
  <S11:Header>  
    ...  
    <wsa:Action xmlns:wsa="...">  
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Cancel  
    </wsa:Action>  
    ...  
    <wsse:Security>  
      <wsc:SecurityContextToken wsu:Id="sct">  
        <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>  
      </wsc:SecurityContextToken>  
      <ds:Signature xmlns:ds="..." Id="sig1">  
        ...signature over body and key headers using #sct...  
      </ds:Signature>  
    </wsse:Security>  
    ...  
  </S11:Header>  
  <S11:Body wsu:Id="req">  
    <wst:RequestSecurityToken>  
      <wst:RequestType>  
        http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel  
      </wst:RequestType>  
      <wst:CancelTarget>  
        <wsse:SecurityTokenReference>  
          <wsse:Reference URI="#sct"/>  
        </wsse:SecurityTokenReference>  
      </wst:CancelTarget>  
    </wst:RequestSecurityToken>
```

731
732

```
</S11:Body>  
</S11:Envelope>
```

733

734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749

```
<S11:Envelope xmlns:S11="..." xmlns:wst="..." >  
  <S11:Header>  
    ...  
    <wsa:Action xmlns:wsa="...">  
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Cancel  
    </wsa:Action>  
    ...  
  </S11:Header>  
  <S11:Body>  
    <wst:RequestSecurityTokenResponseCollection>  
      <wst:RequestSecurityTokenResponse>  
        <wst:RequestedTokenCancelled/>  
      </wst:RequestSecurityTokenResponse>  
    </wst:RequestSecurityTokenResponseCollection>  
  </S11:Body>  
</S11:Envelope>
```

7 Deriving Keys

750

751 A security context token implies or contains a shared secret. This secret MAY be used for signing and/or
752 encrypting messages, but it is RECOMMENDED that derived keys be used for signing and encrypting
753 messages associated only with the security context.

754

755 Using a common secret, parties may define different key derivations to use. For example, four keys may
756 be derived so that two parties can sign and encrypt using separate keys. In order to keep the keys fresh
757 (prevent providing too much data for analysis), subsequent derivations may be used. We introduce the
758 `<wsc:DerivedKeyToken>` token as a mechanism for indicating which derivation is being used within a
759 given message.

760

761 The derived key mechanism can use different algorithms for deriving keys. The algorithm is expressed
762 using a URI. This specification defines one such algorithm.

763

764 As well, while presented here using security context tokens, the `<wsc:DerivedKeyToken>` token can
765 be used to derive keys from any security token that has a shared secret, key, or key material.

766

767 We use a subset of the mechanism defined for TLS in RFC 2246. Specifically, we use the P_SHA-1
768 function to generate a sequence of bytes that can be used to generate security keys. We refer to this
769 algorithm as:

770

771

```
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk/p_shal
```

772

773 This function is used with three values – *secret*, *label*, and *seed*. The secret is the shared secret that is
774 exchanged (note that if two secrets were securely exchanged, possible as part of an initial exchange,
775 they are concatenated in the order they were sent/received). Secrets are processed as octets
776 representing their binary value (value prior to encoding). The label is the concatenation of the client's
777 label and the service's label. These labels can be discovered in each party's policy (or specifically within
778 a `<wsc:DerivedKeyToken>` token). Labels are processed as UTF-8 encoded octets. If either isn't
779 specified in the policy, then a default value of "WS-SecureConversation" (represented as UTF-8 octets) is
780 used. The seed is the concatenation of nonce values (if multiple were exchanged) that were exchanged
781 (initiator + receiver). The nonce is processed as a binary octet sequence (the value prior to base64
782 encoding). The nonce seed is required, and MUST be generated by one or more of the communicating
783 parties. The P_SHA-1 function has two parameters – *secret* and *value*. We concatenate the *label* and
784 the *seed* to create the *value*. That is:

785

```
P_SHA1 (secret, label + seed)
```

786

787 At this point, both parties can use the P_SHA-1 function to generate shared keys as needed. For this
788 protocol, we don't define explicit derivation uses.

789

790 The `<wsc:DerivedKeyToken>` element is used to indicate that the key for a specific reference is
791 generated from the function. This is so that explicit security tokens, secrets, or key material need not be
792 exchanged as often thereby increasing efficiency and overall scalability. However, parties MUST

793 mutually agree on specific derivations (e.g. the first 128 bits is the client's signature key, the next 128 bits
794 in the client's encryption key, and so on). The policy presents a method for specifying this information.
795 The RECOMMENDED approach is to use separate nonces and have independently generated keys for
796 signing and encrypting in each direction. Furthermore, it is RECOMMENDED that new keys be derived
797 for each message (i.e., previous nonces are not re-used).

798

799 Once the parties determine a shared secret to use as the basis of a key generation sequence, an initial
800 key is generated using this sequence. When a new key is required, a new `<wsc:DerivedKeyToken>`
801 may be passed referencing the previously generated key. The recipient then knows to use the sequence
802 to generate a new key, which will match that specified in the security token. If both parties pre-agree on
803 key sequencing, then additional token exchanges are not required.

804

805 For keys derived using a shared secret from a security context, the
806 `<wsse:SecurityTokenReference>` element SHOULD be used to reference the
807 `<wsc:SecurityContextToken>`. Basically, a signature or encryption references a
808 `<wsc:DerivedKeyToken>` in the `<wsse:Security>` header that, in turn, references the
809 `<wsc:SecurityContextToken>`.

810

811 Derived keys are expressed as security tokens. The following URI is used to represent the token type:

812

```
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk
```

813

814 The derived key token does not support references using key identifiers or key names. All references
815 MUST use an ID (to a `wsu:id` attribute) or a URI reference to the `<wsc:Identifier>` element in the
816 SCT.

817 7.1 Syntax

818 The following illustrates the syntax for `<wsc:DerivedKeyToken>` is as follows:

819

```
820 <wsc:DerivedKeyToken wsu:Id="..." Algorithm="..." xmlns:wsc="..."  
xmlns:wsse="..." xmlns:wsu="...">  
821 <wsse:SecurityTokenReference>...</wsse:SecurityTokenReference>  
822 <wsc:Properties>...</wsc:Properties>  
823 <wsc:Generation>...</wsc:Generation>  
824 <wsc:Offset>...</wsc:Offset>  
825 <wsc:Length>...</wsc:Length>  
826 <wsc:Label>...</wsc:Label>  
827 <wsc:Nonce>...</wsc:Nonce>  
828 </wsc:DerivedKeyToken>
```

829

830 The following describes the attributes and tags listed in the schema overview above:

831 `/wsc:DerivedKeyToken`

832 This specifies a key that is derived from a shared secret.

833 `/wsc:DerivedKeyToken/@wsu:Id`

834 This optional attribute specifies an XML ID that can be used locally to reference this element.

835 `/wsc:DerivedKeyToken/@Algorithm`

836 This optional URI attribute specifies key derivation algorithm to use. This specification predefines
837 the P_SHA1 algorithm described above. If this attribute isn't specified, this algorithm is assumed.

838 /wsc:DerivedKeyToken/wsse:SecurityTokenReference

839 This optional element is used to specify security context token, security token, or shared
840 key/secret used for the derivation. If not specified, it is assumed that the recipient can determine
841 the shared key from the message context. If the context cannot be determined, then a fault such
842 as `wsc:UnknownDerivationSource` should be raised.

843 /wsc:DerivedKeyToken/wsc:Properties

844 This optional element allows metadata to be associated with this derived key. For example, if the
845 `<wsc:Name>` property is defined, this derived key is given a URI name that can then be used as
846 the source for other derived keys. The `<wsc:Nonce>` and `<wsc:Label>` elements can be
847 specified as properties and indicate the nonce and label to use (defaults) for all keys derived from
848 this key.

849 /wsc:DerivedKeyToken/wsc:Properties/wsc:Name

850 This optional element is used to give this derived key a URI name that can then be used as the
851 source for other derived keys.

852 /wsc:DerivedKeyToken/wsc:Properties/wsc:Label

853 This optional element defines a label to use for all keys derived from this key. See
854 `/wsc:DerivedKeyToken/wsc:Label` defined below.

855 /wsc:DerivedKeyToken/wsc:Properties/wsc:Nonce

856 This optional element defines a label to use for all keys derived from this key. See
857 `/wsc:DerivedKeyToken/wsc:Nonce` defined below.

858 /wsc:DerivedKeyToken/wsc:Properties/{any}

859 This is an extensibility mechanism to allow additional elements (arbitrary content) to be used.

860 /wsc:DerivedKeyToken/wsc:Generation

861 If fixed-size keys (generations) are being generated, then this optional element can be used to
862 specify which generation of the key to use. The value of this element is an unsigned long value
863 indicating the generation number to use (beginning with zero). This element MUST NOT be used
864 if the `<wsc:Offset>` element is specified. Specifying this element is equivalent to specifying the
865 `<wsc:Offset>` and `<wsc:Length>` elements having multiplied out the values. That is, $\text{offset} =$
866 $(\text{generation}) * \text{fixed_size}$ and $\text{length} = \text{fixed_size}$.

867 /wsc:DerivedKeyToken/wsc:Offset

868 If fixed-size keys are not being generated, then the `<wsc:Offset>` and `<wsc:Length>`
869 elements indicate where in the byte stream to find the generated key. This specifies the ordering
870 (in bytes) of the generated output. The value of this optional element is an unsigned long value
871 indicating the byte position (starting at 0). For example, 0 indicates the first byte of output and 16
872 indicates the 17th byte of generated output. This element MUST NOT be used if the
873 `<wsc:Generation>` element is specified. It should be noted that not all algorithms will support
874 the `<wsc:Offset>` and `<wsc:Length>` elements.

875 /wsc:DerivedKeyToken/wsc:Length

876 This element specifies the length (in bytes) of the derived key. This optional element can be
877 specified in conjunction with `<wsc:Offset>` or `<wsc:Generation>`. If this isn't specified, it is
878 assumed that the recipient knows the key size to use. The value of this element is an unsigned
879 long value indicating the size of the key in bytes (e.g., 16).

880 /wsc:DerivedKeyToken/wsc:Label

881 The label can be specified within a `<wsc:DerivedKeyToken>` using the `wsc:Label` element. If the
882 label isn't specified then a default value of "WS-SecureConversationWS-SecureConversation"
883 (represented as UTF-8 octets) is used. Labels are processed as UTF-8 encoded octets..

884 /wsc:DerivedKeyToken/wsc:Nonce

885 If specified, this optional element specifies a base64 encoded nonce that is used in the key
886 derivation function for this derived key. If this isn't specified, it is assumed that the recipient
887 knows the nonce to use. Note that once a nonce is used for a derivation sequence, the same
888 nonce SHOULD be used for all subsequent derivations.

889

890 If additional information is not specified (such as explicit elements or policy), then the following defaults
891 apply:

892 The offset is 0

893 The length is 32 bytes (256 bits)

894

895 It is RECOMMENDED that separate derived keys be used to strengthen the cryptography. If multiple keys
896 are used, then care should be taken not to derive too many times and risk key attacks.

897 7.2 Examples

898 The following example illustrates a message sent using two derived keys, one for signing and one for
899 encrypting:

```
900 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsc="..."  
901   xmlns:xenc="..." xmlns:ds="..."  
902   <S11:Header>  
903     <wsse:Security>  
904       <wsc:SecurityContextToken wsu:Id="ctx2">  
905         <wsc:Identifier>uuid:...UUID2...</wsc:Identifier>  
906       </wsc:SecurityContextToken>  
907       <wsc:DerivedKeyToken wsu:Id="dk2">  
908         <wsse:SecurityTokenReference>  
909           <wsse:Reference URI="#ctx2"/>  
910         </wsse:SecurityTokenReference>  
911         <wsc:Nonce>KJHFRE...</wsc:Nonce>  
912       </wsc:DerivedKeyToken>  
913     <xenc:ReferenceList>  
914       ...  
915     <ds:KeyInfo>  
916       <wsse:SecurityTokenReference>  
917         <wsse:Reference URI="#dk2"/>  
918       </wsse:SecurityTokenReference>  
919     </ds:KeyInfo>  
920     ...  
921   </xenc:ReferenceList>  
922   <wsc:SecurityContextToken wsu:Id="ctx1">  
923     <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>  
924   </wsc:SecurityContextToken>  
925   <wsc:DerivedKeyToken wsu:Id="dk1">  
926     <wsse:SecurityTokenReference>  
927       <wsse:Reference URI="#ctx1"/>  
928     </wsse:SecurityTokenReference>  
929     <wsc:Nonce>KJHFRE...</wsc:Nonce>  
930   </wsc:DerivedKeyToken>  
931   <xenc:ReferenceList>  
932     ...  
933   <ds:KeyInfo>  
934     <wsse:SecurityTokenReference>  
935       <wsse:Reference URI="#dk1"/>  
936     </wsse:SecurityTokenReference>  
937   </ds:KeyInfo>  
938   ...
```

```
939         </xenc:ReferenceList>
940     </wsse:Security>
941     ...
942 </S11:Header>
943 <S11:Body>
944     ...
945 </S11:Body>
946 </S11:Envelope>
```

947

948 The following illustrates the syntax for a derived key based on the 3rd generation of the shared key
949 identified in the specified security context:

```
950 <wsc:DerivedKeyToken xmlns:wsc="..." xmlns:wsse="...">
951   <wsse:SecurityTokenReference>
952     <wsse:Reference URI="#ctx1"/>
953   </wsse:SecurityTokenReference>
954   <wsc:Generation>2</wsc:Generation>
955 </wsc:DerivedKeyToken>
```

956

957 The following illustrates the syntax for a derived key based on the 1st generation of a key derived from an
958 existing derived key (4th generation):

```
959 <wsc:DerivedKeyToken xmlns:wsc="...">
960   <wsc:Properties>
961     <wsc:Name>.../derivedKeySource</wsc:Name>
962     <wsc:Label>NewLabel</wsc:Label>
963     <wsc:Nonce>FHFE...</wsc:Nonce>
964   </wsc:Properties>
965   <wsc:Generation>3</wsc:Generation>
966 </wsc:DerivedKeyToken>
```

967

```
968 <wsc:DerivedKeyToken wsu:Id="newKey" xmlns:wsc="..." xmlns:wsse="..." >
969   <wsse:SecurityTokenReference>
970     <wsse:Reference URI=".../derivedKeySource"/>
971   </wsse:SecurityTokenReference>
972   <wsc:Generation>0</wsc:Generation>
973 </wsc:DerivedKeyToken>
```

974

975 In the example above we have named a derived key so that other keys can be derived from it. To do this
976 we use the `<wsc:Properties>` element name tag to assign a global name attribute. Note that in this
977 example, the ID attribute could have been used to name the base derived key if we didn't want it to be a
978 globally named resource. We have also included the `<wsc:Label>` and `<wsc:Nonce>` elements as
979 metadata properties indicating how to derive sequences of this derivation.

980 7.3 Implied Derived Keys

981 This specification also defines a shortcut mechanism for referencing certain types of derived keys.
982 Specifically, a `@wsc:Nonce` attribute can also be added to the security token reference (STR) defined in
983 the [WS-Security] specification. When present, it indicates that the key is not in the referenced token, but
984 is a key derived from the referenced token's key/secret. The `@wsc:Length` attribute can be used in
985 conjunction with `@wsc:Nonce` in the security token reference (STR) to indicate the length of the derived
986 key. The value of this attribute is an unsigned long value indicating the size of the key in bytes. If this
987 attribute isn't specified, the default derived key length value is 32.

988

989 Consequently, the following two illustrations are functionally equivalent:

```
990 <wsse:Security xmlns:wsc="..." xmlns:wsse="..." xmlns:xx="..."
991 xmlns:ds="..." xmlns:wsu="...">
992 <xx:MyToken wsu:Id="base">...</xx:MyToken>
993 <wsc:DerivedKeyToken wsu:Id="newKey">
994 <wsse:SecurityTokenReference>
995 <wsse:Reference URI="#base"/>
996 </wsse:SecurityTokenReference>
997 <wsc:Nonce>...</wsc:Nonce>
998 </wsc:DerivedKeyToken>
999 <ds:Signature>
1000 ...
1001 <ds:KeyInfo>
1002 <wsse:SecurityTokenReference>
1003 <wsse:Reference URI="#newKey"/>
1004 </wsse:SecurityTokenReference>
1005 </ds:KeyInfo>
1006 </ds:Signature>
1007 </wsse:Security>
```

1008

1009 This is functionally equivalent to the following:

```
1010 <wsse:Security xmlns:wsc="..." xmlns:wsse="..." xmlns:xx="..."
1011 xmlns:ds="..." xmlns:wsu="...">
1012 <xx:MyToken wsu:Id="base">...</xx:MyToken>
1013 <ds:Signature>
1014 ...
1015 <ds:KeyInfo>
1016 <wsse:SecurityTokenReference wsc:Nonce="...">
1017 <wsse:Reference URI="#base"/>
1018 </wsse:SecurityTokenReference>
1019 </ds:KeyInfo>
1020 </ds:Signature>
1021 </wsse:Security>
```

8 Associating a Security Context

1022

1023 For a variety of reasons it may be necessary to reference a Security Context Token. These references
1024 can be broken into two general categories: references from within the `<wsse:Security>` element to a
1025 token also within the `<wsse:Security>` element, generally used to indicate the key used in a signature
1026 or encryption operation and references from other parts of the SOAP envelope, for example to specify a
1027 token to be used in some particular way. References within the `<wsse:Security>` element can further
1028 be divided into reference to an SCT found within the message and references to a SCT not present in the
1029 message.

1030

1031 The Security Context Token does not support references to it using key identifiers or key names. All
1032 references **MUST** either use an ID (to a `wsu:Id` attribute) or a `<wsse:Reference>` to the
1033 `<wsc:Identifier>` element.

1034

1035 References using an ID are message-specific. References using the `<wsc:Identifier>` element value
1036 are message independent.

1037

1038 If the SCT is referenced from within the `<wsse:Security>` element or from an RST or RSTR, it is
1039 **RECOMMENDED** that these references be message independent, but these references **MAY** be
1040 message-specific.

1041

1042 When an SCT located in the `<wsse:Security>` element is referenced from outside the
1043 `<wsse:Security>` element, a message independent referencing mechanisms **MUST** be used, to
1044 enable a cleanly layered processing model unless there is a prior agreement between the involved parties
1045 to use message-specific referencing mechanism.

1046

1047 When an SCT is referenced from within the `<wsse:Security>` element, but the SCT is not present in
1048 the message, (presumably because it was transmitted in a previous message) a message independent
1049 referencing mechanism **MUST** be used.

1050

1051 The following example illustrates associating a specific security context with an action.

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
  xmlns:wsc="...">
  <S11:Header>
    ...
    <wsse:Security>
      <wsc:SecurityContextToken wsu:Id="sct1">
        <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
      </wsc:SecurityContextToken>
      <ds:Signature xmlns:ds="...">
        ...signature over body and key headers using #sct1...
      </ds:Signature>
      <wsc:SecurityContextToken wsu:Id="sct2">
        <wsc:Identifier>uuid:...UUID2...</wsc:Identifier>
      </wsc:SecurityContextToken>
      <ds:Signature xmlns:ds="...">
        ...signature over body and key headers using #sct2...
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  ...
</S11:Envelope>
```

```
1071 </S11:Header>
1072 <S11:Body wsu:Id="req">
1073   <xx:Custom xmlns:xx="http://example.com/custom" xmlns:wsse="...">
1074     ...
1075     <wsse:SecurityTokenReference>
1076       <wsse:Reference URI="#sct2"/>
1077     </wsse:SecurityTokenReference>
1078   </xx:Custom>
1079 </S11:Body>
1080 </S11:Envelope>
```

1081

9 Error Handling

1082

There are many circumstances where an *error* can occur while processing security information. Errors use the SOAP Fault mechanism. Note that the reason text provided below is RECOMMENDED, but alternative text MAY be provided if more descriptive or preferred by the implementation. The tables below are defined in terms of SOAP 1.1. For SOAP 1.2, the Fault/Code/Value is env:Sender (as defined in SOAP 1.2) and the Fault/Code/Subcode/Value is the *faultcode* below and the Fault/Reason/Text is the *faultstring* below. It should be noted that profiles MAY provide second-level details fields, but they should be careful not to introduce security vulnerabilities when doing so (e.g. by providing too detailed information).

1083

1084

1085

1086

1087

1088

1089

Error that occurred (faultstring)	Fault code (faultcode)
The requested context elements are insufficient or unsupported.	wsc:BadContextToken
Not all of the values associated with the SCT are supported.	wsc:UnsupportedContextToken
The specified source for the derivation is unknown.	wsc:UnknownDerivationSource
The provided context token has expired	wsc:RenewNeeded
The specified context token could not be renewed.	wsc:UnableToRenew

1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115

10 Security Considerations

As stated in the Goals section of this document, this specification is meant to provide extensible framework and flexible syntax, with which one could implement various security mechanisms. This framework and syntax by itself *does not provide any guarantee of security*. When implementing and using this framework and syntax, one must make every effort to ensure that the result is not vulnerable to any one of a wide range of attacks.

It is not feasible to provide a comprehensive list of security considerations for such an extensible set of mechanisms. A complete security analysis must be conducted on specific solutions based on this specification. Below we illustrate some of the security concerns that often come up with protocols of this type, but we stress that this *is not an exhaustive list of concerns*.

It is critical that all relevant elements of a message be included in signatures. As well, the signatures for security context establishment must include a timestamp, nonce, or sequence number depending on the degree of replay prevention required. Security context establishment should include full policies to prevent possible attacks (e.g. downgrading attacks).

Authenticating services are susceptible to denial of service attacks. Care should be taken to mitigate such attacks as is warranted by the service.

There are many other security concerns that one may need to consider in security protocols. The list above should not be used as a "check list" instead of a comprehensive security analysis.

In addition to the consideration identified here, readers should also review the security considerations in [\[WS-Security\]](#) and [\[WS-Trust\]](#).

1116

A. Sample Usages

1117 This non-normative appendix illustrates several sample usage patterns of [WS-Trust] and this document.
1118 Specifically, it illustrates different patterns that could be used to parallel, at an end-to-end message level,
1119 the selected TLS/SSL scenarios. This is not intended to be the definitive method for the scenarios, nor is
1120 it fully inclusive. Its purpose is simply to illustrate, in a context familiar to readers, how this specification
1121 might be used.

1122 The following sections are based on a scenario where the client wishes to authenticate the server prior to
1123 sharing any of its own credentials.

1124

1125 It should be noted that the following sample usages are illustrative; any implementation of the examples
1126 illustrated below should be carefully reviewed for potential security attacks. For example, multi-leg
1127 exchanges such as those below should be careful to prevent man-in-the-middle attacks or downgrade
1128 attacks. It may be desirable to use running hashes as challenges that are signed or a similar mechanism
1129 to ensure continuity of the exchange.

1130 The examples below assume that both parties understand the appropriate security policies in use and
1131 can correctly construct signatures and encryption that the other party can process.

A.1 Anonymous SCT

1132 In this scenario the requestor wishes to remain anonymous while authenticating the recipient and
1133 establishing an SCT for secure communication.

1134

1135 This scenario assumes that the requestor has a key for the recipient. If this isn't the case, they can use
1136 [WS-MEX] or the mechanisms described in a later section or obtain one from another security token
1137 service.
1138

1139

1140 There are two basic patterns that can apply, which only vary slightly. The first is as follows:

- 1141 1. The requestor sends an RST to the recipient requesting an SCT. The request contains key
1142 material encrypted for the recipient. The request is not authenticated.
- 1143 2. The recipient, if it accepts such requests, returns an RSTRC with one or more RSTRs with the
1144 SCT as the requested token and does not return any proof information indicating that the
1145 requestor's key is the proof.

1146 A slight variation on this is as follows:

- 1147 1. The requestor sends an RST to the recipient requesting an SCT. The request contains key
1148 material encrypted for the recipient. The request is not authenticated.
- 1149 2. The recipient, if it accepts such requests, returns an RSTRC with one or more RSTR and with the
1150 SCT as the requested token and returns its own key material encrypted using the requestor's key.

1151

1152 Another slight variation is to return a new key encrypted using the requestor's provided key.

1153 It should be noted that the variations that involve encrypting data using the requestor's key material might
1154 be subject to certain types of key attacks.

1155 Yet another approach is to establish a secure channel (e.g. TLS/SSL IP/Sec) between the requestor and
1156 the recipient. Key material can then safely flow in either direction. In some circumstances, this provides
1157 greater protection than the approach above when returning key information to the requestor.

1158 **A.2 Mutual Authentication SCT**

1159 In this scenario the requestor is willing to authenticate, but wants the recipient to authenticate first. The
1160 following steps outline the message flow:

- 1161 1. The requestor sends an RST requesting an SCT. The request contains key material encrypted
1162 for the recipient. The request is not authenticated.
- 1163 2. The recipient returns an RSTRC with one or more RSTRs including a challenge for the requestor.
1164 The RSTRC is secured by the recipient so that the requestor can authenticate it.
- 1165 3. The requestor, after authenticating the recipient's RSTRC, sends an RSTRC responding to the
1166 challenge.
- 1167 4. The recipient, after authenticating the requestor's RSTRC, sends a secured RSTRC containing
1168 the token and either proof information or partial key material (depending on whether or not the
1169 requestor provided key material).

1170

1171 Another variation exists where step 1 includes a specific challenge for the service. Depending on the
1172 type of challenge used this may not be necessary because the message may contain enough entropy to
1173 ensure a fresh response from the recipient.

1174

1175 In other variations the requestor doesn't include key information until step 3 so that it can first verify the
1176 signature of the recipient in step 2.

1177 **B. Token Discovery Using RST/RSTR**

1178 If the recipient's security token is not known, the RST/RSTR mechanism can still be used. The following
1179 example illustrates one possible sequence of messages:

- 1180 1. The requestor sends an RST requesting an SCT. This request does not contain any key
1181 material, nor is the request authenticated.
- 1182 2. The recipient sends an RSTRC with one or more RSTRs to the requestor with an embedded
1183 challenge. The RSTRC is secured by the recipient so that the requestor can authenticate it.
- 1184 3. The requestor sends an RSTRC to the recipient and includes key information protected for the
1185 recipient. This request may or may not be secured depending on whether or not the request is
1186 anonymous.
- 1187 4. The final issuance step depends on the exact scenario. Any of the final legs from above might be
1188 used.

1189
1190 Note that step 1 might include a challenge for the recipient. Please refer to the comment in the previous
1191 section on this scenario.

1192 Also note that in response to step 1 the recipient might issue a fault secured with [[WS-Security](#)] providing
1193 the requestor with information about the recipient's security token.

1194

C. Acknowledgements

1195 The following individuals have participated in the creation of this specification and are gratefully
1196 acknowledged.

1197

1198 **Original Authors of the initial contribution:**

1199 Steve Anderson, OpenNetwork

1200 Jeff Bohren, OpenNetwork

1201 Toufic Boubez, Layer 7

1202 Marc Chanliau, Computer Associates

1203 Giovanni Della-Libera, Microsoft

1204 Brendan Dixon, Microsoft

1205 Praerit Garg, Microsoft

1206 Martin Gudgin (Editor), Microsoft

1207 Satoshi Hada, IBM

1208 Phillip Hallam-Baker, VeriSign

1209 Maryann Hondo, IBM

1210 Chris Kaler, Microsoft

1211 Hal Lockhart, BEA

1212 Robin Martherus, Oblix

1213 Hiroshi Maruyama, IBM

1214 Anthony Nadalin (Editor), IBM

1215 Nataraj Nagaratnam, IBM

1216 Andrew Nash, Reactivity

1217 Rob Philpott, RSA Security

1218 Darren Platt, Ping Identity

1219 Hemma Prafullchandra, VeriSign

1220 Maneesh Sahu, Actional

1221 John Shewchuk, Microsoft

1222 Dan Simon, Microsoft

1223 Davanum Srinivas, Computer Associates

1224 Elliot Waingold, Microsoft

1225 David Waite, Ping Identity

1226 Doug Walter, Microsoft

1227 Riaz Zolfonoon, RSA Security

1228

1229 **Original Acknowledgements of the initial contribution:**

1230 Paula Austel, IBM

1231 Keith Ballinger, Microsoft

1232 John Brezak, Microsoft

1233 Tony Cowan, IBM

1234 HongMei Ge, Microsoft

1235 Slava Kavsan, RSA Security

1236 Scott Konersmann, Microsoft

1237 Leo Laferriere, Computer Associates

1238 Paul Leach, Microsoft

1239 Richard Levinson, Computer Associates

1240 John Linn, RSA Security

1241 Michael McIntosh, IBM

1242 Steve Millet, Microsoft
1243 Birgit Pfitzmann, IBM
1244 Fumiko Satoh, IBM
1245 Keith Stobie, Microsoft
1246 T.R. Vishwanath, Microsoft
1247 Richard Ward, Microsoft
1248 Hervey Wilson, Microsoft

1249

1250 **TC Members during the development of this specification:**

1251 Don Adams, Tibco Software Inc.
1252 Jan Alexander, Microsoft Corporation
1253 Steve Anderson, BMC Software
1254 Donal Arundel, IONA Technologies
1255 Howard Bae, Oracle Corporation
1256 Abbie Barbir, Nortel Networks Limited
1257 Charlton Barreto, Adobe Systems
1258 Mighael Botha, Software AG, Inc.
1259 Toufic Boubez, Layer 7 Technologies Inc.
1260 Norman Brickman, Mitre Corporation
1261 Melissa Brumfield, Booz Allen Hamilton
1262 Lloyd Burch, Novell
1263 Scott Cantor, Internet2
1264 Greg Carpenter, Microsoft Corporation
1265 Steve Carter, Novell
1266 Ching-Yun (C.Y.) Chao, IBM
1267 Martin Chapman, Oracle Corporation
1268 Kate Cherry, Lockheed Martin
1269 Henry (Hyenvui) Chung, IBM
1270 Luc Clement, Systinet Corp.
1271 Paul Cotton, Microsoft Corporation
1272 Glen Daniels, Sonic Software Corp.
1273 Peter Davis, Neustar, Inc.
1274 Martijn de Boer, SAP AG
1275 Werner Dittmann, Siemens AG
1276 Abdeslem DJAOUI, Associate Member
1277 Fred Dushin, IONA Technologies
1278 Petr Dvorak, Systinet Corp.
1279 Colleen Evans, Microsoft Corporation
1280 Ruchith Fernando, WSO2
1281 Mark Fussell, Microsoft Corporation
1282 Vijay Gajjala, Microsoft Corporation
1283 Marc Goodner, Microsoft Corporation
1284 Hans Granqvist, VeriSign
1285 Martin Gudgin, Microsoft Corporation
1286 Tony Gullotta, SOA Software Inc.
1287 Jiandong Guo, Sun Microsystems
1288 Phillip Hallam-Baker, VeriSign
1289 Patrick Harding, Ping Identity Corporation
1290 Heather Hinton, IBM
1291 Frederick Hirsch, Nokia Corporation
1292 Jeff Hodges, Neustar, Inc.
1293 Will Hopkins, BEA Systems, Inc.
1294 Alex Hristov, Otecia Incorporated
1295 John Hughes, Associate Member
1296 Diane Jordan, IBM
1297 Venugopal K, Sun Microsystems

1298 Chris Kaler, Microsoft Corporation
1299 Dana Kaufman, Forum Systems, Inc.
1300 Paul Knight, Nortel Networks Limited
1301 Ramanathan Krishnamurthy, IONA Technologies
1302 Christopher Kurt, Microsoft Corporation
1303 Kelvin Lawrence, IBM
1304 Hubert Le Van Gong, Sun Microsystems
1305 Jong Lee, BEA Systems, Inc.
1306 Rich Levinson, Oracle Corporation
1307 Tommy Lindberg, Associate Member
1308 Mark Little, JBoss Inc.
1309 Hal Lockhart, BEA Systems, Inc.
1310 Mike Lyons, Layer 7 Technologies Inc.
1311 Eve Maler, Sun Microsystems
1312 Ashok Malhotra, Oracle Corporation
1313 Anand Mani, CrimsonLogic Pte Ltd
1314 Jonathan Marsh, Microsoft Corporation
1315 Robin Martherus, Oracle Corporation
1316 Miko Matsumura, Infravio, Inc.
1317 Gary McAfee, IBM
1318 Michael McIntosh, IBM
1319 John Merrells, Sxip Networks SRL
1320 Jeff Mischkinsky, Oracle Corporation
1321 Prateek Mishra, Oracle Corporation
1322 Bob Morgan, Internet2
1323 Vamsi Motukuru, Oracle Corporation
1324 Raajmohan Na, EDS
1325 Anthony Nadalin, IBM
1326 Andrew Nash, Reactivity, Inc.
1327 Eric Newcomer, IONA Technologies
1328 Duane Nickull, Adobe Systems
1329 Toshihiro Nishimura, Fujitsu Limited
1330 Rob Philpott, RSA Security
1331 Denis Pilipchuk, BEA Systems, Inc.
1332 Darren Platt, Ping Identity Corporation
1333 Martin Raepple, SAP AG
1334 Nick Ragouzis, Associate Member
1335 Prakash Reddy, CA
1336 Alain Regnier, Ricoh Company, Ltd.
1337 Irving Reid, Hewlett-Packard
1338 Bruce Rich, IBM
1339 Tom Rutt, Fujitsu Limited
1340 Maneesh Sahu, Actional Corporation
1341 Frank Siebenlist, Argonne National Laboratory
1342 Joe Smith, Apani Networks
1343 Davanum Srinivas, WSO2
1344 Yakov Sverdlov, CA
1345 Gene Thurston, AmberPoint
1346 Victor Valle, IBM
1347 Asir Vedamuthu, Microsoft Corporation
1348 Greg Whitehead, Hewlett-Packard
1349 Ron Williams, IBM
1350 Corinna Witt, BEA Systems, Inc.
1351 Kyle Young, Microsoft Corporation