



WS-SecureConversation 1.4

OASIS Committee Draft 03

12 November 2008

Specification URIs:

This Version:

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-03.doc> (Authoritative)
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-03.pdf>
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-03.html>

Previous Version:

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-02.doc> (Authoritative)
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-02.pdf>
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-02.html>

Latest Version:

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.doc>
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.pdf>
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html>

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

Related work:

NA

Declared XML namespace(s):

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512>

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® 1993–2008. All Rights Reserved.

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

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

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

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

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

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

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

The name "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.....	5
1.1	Goals and Non-Goals.....	5
1.2	Requirements.....	5
1.3	Namespace.....	5
1.4	Schema File.....	6
1.5	Terminology.....	6
1.5.1	Notational Conventions.....	7
1.6	Normative References.....	8
1.7	Non-Normative References.....	9
2	Security Context Token (SCT).....	10
3	Establishing Security Contexts.....	13
3.1	SCT Binding of WS-Trust.....	14
3.2	SCT Request Example without Target Scope.....	14
3.3	SCT Request Example with Target Scope.....	15
3.4	SCT Propagation Example.....	17
4	Amending Contexts.....	18
5	Renewing Contexts.....	20
6	Canceling Contexts.....	22
7	Deriving Keys.....	24
7.1	Syntax.....	25
7.2	Examples.....	27
7.3	Implied Derived Keys.....	28
8	Associating a Security Context.....	30
9	Error Handling.....	32
10	Security Considerations.....	33
11	Conformance.....	34
A.	Sample Usages.....	35
A.1	Anonymous SCT.....	35
A.2	Mutual Authentication SCT.....	36
B.	Token Discovery Using RST/RSTR.....	37
C.	Acknowledgements.....	38

1 Introduction

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

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

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

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

1.1 Goals and Non-Goals

The primary goals of this specification are:

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

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

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

1.2 Requirements

The following list identifies the key driving requirements:

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

1.3 Namespace

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

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

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

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]

37 1.4 Schema File

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

39 [http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-](http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3.xsd)
 40 [secureconversation-1.3.xsd](http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3.xsd)

41
 42 In this document, reference is made to the `wsu:Id` attribute in the utility schema. These were added to
 43 the utility schema with the intent that other specifications requiring such an ID or timestamp could
 44 reference it (as is done here).

45 1.5 Terminology

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

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

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

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

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

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

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

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

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

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

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

79 **1.5.1 Notational Conventions**

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

83

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

86

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

- 88 • The syntax appears as an XML instance, but values in italics indicate data types instead of literal
89 values.
- 90 • Characters are appended to elements and attributes to indicate cardinality:
 - 91 ○ "?" (0 or 1)
 - 92 ○ "*" (0 or more)
 - 93 ○ "+" (1 or more)
- 94 • The character "|" is used to indicate a choice between alternatives.
- 95 • The characters "(" and ")" are used to indicate that contained items are to be treated as a group
96 with respect to cardinality or choice.
- 97 • The characters "[" and "]" are used to call out references and property names.
- 98 • Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be
99 added at the indicated extension points but MUST NOT contradict the semantics of the parent

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

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

105

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

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

114

115 In this document reference is made to the `wsu:Id` attribute and the `wsu:Created` and `wsu:Expires`
116 elements in a utility schema ([http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
117 1.0.xsd](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 1.6 Normative References

- 122 **[RFC2119]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC
123 2119, Harvard University, March 1997.
124 <http://www.ietf.org/rfc/rfc2119.txt> .
- 125 **[RFC2246]** IETF Standard, "The TLS Protocol", January 1999.
126 <http://www.ietf.org/rfc/rfc2246.txt>
- 127 **[SOAP]** W3C Note, "SOAP: Simple Object Access Protocol 1.1", 08 May 2000.
128 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
- 129 **[SOAP12]** W3C Recommendation, "SOAP 1.2 Part 1: Messaging Framework", 24 June
130 2003.
131 <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>
- 132 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI):
133 Generic Syntax", RFC 3986, MIT/LCS, Day Software, Adobe Systems, January
134 2005.
135 <http://www.ietf.org/rfc/rfc3986.txt>
- 136 **[WS-Addressing]** W3C Recommendation, "Web Services Addressing (WS-Addressing)", 9 May
137 2006.
138 <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509>.
- 139 **[WS-Security]** OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.0
140 (WS-Security 2004)", March 2004.
141 [http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
142 security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
- 143 OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.1
144 (WS-Security 2004)", February 2006.
145 [http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-
146 SOAPMessageSecurity.pdf](http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)
- 147 **[WS-Trust]** OASIS Committee Draft, "WS-Trust 1.4", 2008
148 <http://docs.oasis-open.org/ws-sx/ws-trust/200802>

149 **[XML-Encrypt]** W3C Recommendation, "XML Encryption Syntax and Processing", 10 December
150 2002.
151 <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>.
152 **[XML-Schema1]** W3C Recommendation, "XML Schema Part 1: Structures Second Edition", 28
153 October 2004.
154 <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>.
155 **[XML-Schema2]** W3C Recommendation, "XML Schema Part 2: Datatypes Second Edition", 28
156 October 2004.
157 <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.
158 **[XML-Signature]** W3C Recommendation, "XML-Signature Syntax and Processing", 12 February
159 2002.
160 <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>
161 W3C Recommendation, D. Eastlake et al. XML Signature Syntax and Processing
162 (Second Edition). 10 June 2008.
163 <http://www.w3.org/TR/2008/REC-xmlsig-core-20080610/>
164

165 **1.7 Non-Normative References**

166 **[WS-MEX]** "Web Services Metadata Exchange (WS-MetadataExchange)", BEA, Computer
167 Associates, IBM, Microsoft, SAP, Sun Microsystems, Inc., webMethods,
168 September 2004.
169 **[WS-SecurityPolicy]** OASIS Committee Draft, "WS-SecurityPolicy 1.2", September 2006
170 <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512>
171
172
173

2 Security Context Token (SCT)

174

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

178

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

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

183

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

187

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

190

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

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

199

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

201 `/wsc:SecurityContextToken`

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

203 `/wsc:SecurityContextToken/wsc:Identifier`

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

207 `/wsc:SecurityContextToken/wsc:Instance`

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

213 `/wsc:SecurityContextToken/@wsu:Id`

214 This **OPTIONAL** attribute specifies a string label for this element.

215 `/wsc:SecurityContextToken/@{any}`

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

218 /wsc:SecurityContextToken/{any}

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

220

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

232

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

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

244

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

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

268
269
270
271
272
273

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

274

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

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

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

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

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

285

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

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

290 3 Establishing Security Contexts

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

294

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

304

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

321

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

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

334 3.1 SCT Binding of WS-Trust

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

339

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

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

344

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

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

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

353 3.2 SCT Request Example without Target Scope

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

```
367 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
368   xmlns:wst="..." xmlns:xenc="...">  
369   <S11:Header>  
370     ...  
371     <wsa:Action xmlns:wsa="...">  
372     http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT  
373     </wsa:Action>  
374     ...  
375     <wsse:Security>  
376       <xenc:EncryptedKey>  
377         ...  
378       </xenc:EncryptedKey>  
379       <xenc:EncryptedData Id="encUsernameToken">  
380         .. encrypted username token (whose id is myToken) ...  
381       </xenc:EncryptedData>  
382       <ds:Signature xmlns:ds="...">  
383         ...
```

```

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

```

```

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

```

432 3.3 SCT Request Example with Target Scope

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

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

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

```

440 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."

```

```

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

```

466

```

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

```

498

499 3.4 SCT Propagation Example

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

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

523

4 Amending Contexts

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

530 The following Action URIs are used with this binding:

531
532

```
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
```

533

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

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

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

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

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
570
571
572

```
<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>  
<S11:Body wsu:Id="req">
```

```
573     <wst:RequestSecurityToken>
574         <wst:RequestType>
575             http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
576         </wst:RequestType>
577     </wst:RequestSecurityToken>
578 </S11:Body>
579 </S11:Envelope>
```

580

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

5 Renewing Contexts

601

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

605
606

```
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
```

607

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

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

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

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

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

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

```
622 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
623     xmlns:wst="..." xmlns:wsc="...">
624   <S11:Header>
625     ...
626     <wsa:Action xmlns:wsa="...">
627       http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Renew
628     </wsa:Action>
629     ...
630     <wsse:Security>
631       <xx:CustomToken wsu:Id="cust" xmlns:xx="...">
632         ...
633       </xx:CustomToken>
634       <ds:Signature xmlns:ds="..." Id="sig1">
635         ... signature over body and key headers using #cust...
636       </ds:Signature>
637       <wsc:SecurityContextToken wsu:Id="sct">
638         <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
639       </wsc:SecurityContextToken>
640       <ds:Signature xmlns:ds="..." Id="sig2">
641         ... signature over #sig1 using #sct ...
642       </ds:Signature>
643     </wsse:Security>
644     ...
645   </S11:Header>
646   <S11:Body wsu:Id="req">
647     <wst:RequestSecurityToken>
648       <wst:RequestType>
```

```
649         http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew
650     </wst:RequestType>
651     <wst:RenewTarget>
652         <wsse:SecurityTokenReference>
653             <wsse:Reference URI="uuid:...UUID1..." />
654         </wsse:SecurityTokenReference>
655     </wst:RenewTarget>
656     <wst:Lifetime>...</wst:Lifetime>
657 </wst:RequestSecurityToken>
658 </S11:Body>
659 </S11:Envelope>
```

660

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

6 Canceling Contexts

683

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

687 The following Action URIs are used with this binding:

688
689

```
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
```

690

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

693

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

697

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

699

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

702

703 The following example illustrates canceling a context.

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
731
732

```
<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="uuid:...UUID1..." />  
        </wsse:SecurityTokenReference>  
      </wst:CancelTarget>  
    </wst:RequestSecurityToken>
```

733
734

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

735

736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751

```
<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>
```

752

7 Deriving Keys

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

756

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

762

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

765

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

768

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

772

773

```
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk/p_sha1
```

774

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

787

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

788

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

791

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

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

800

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

806

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

812

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

814

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

815

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

819 7.1 Syntax

820 The following illustrates the syntax for `<wsc:DerivedKeyToken>`:

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

831

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

833 `/wsc:DerivedKeyToken`

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

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

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

837 `/wsc:DerivedKeyToken/@Algorithm`

838 This OPTIONAL URI attribute specifies key derivation algorithm to use. This specification
839 predefines the P_SHA1 algorithm described above. If this attribute isn't specified, this algorithm is
840 assumed.

841 /wsc:DerivedKeyToken/wsse:SecurityTokenReference

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

846 /wsc:DerivedKeyToken/wsc:Properties

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

852 /wsc:DerivedKeyToken/wsc:Properties/wsc:Name

853 This OPTIONAL element is used to give this derived key a URI name that can then be used as
854 the source for other derived keys.

855 /wsc:DerivedKeyToken/wsc:Properties/wsc:Label

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

858 /wsc:DerivedKeyToken/wsc:Properties/wsc:Nonce

859 This OPTIONAL element defines a nonce to use for all keys derived from this key. See
860 /wsc:DerivedKeyToken/wsc:Nonce defined below.

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

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

863 /wsc:DerivedKeyToken/wsc:Generation

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

870 /wsc:DerivedKeyToken/wsc:Offset

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

878 /wsc:DerivedKeyToken/wsc:Length

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

883 /wsc:DerivedKeyToken/wsc:Label

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

887 /wsc:DerivedKeyToken/wsc:Nonce

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

892

893 If additional information is not specified as explicit elements, then the following defaults apply:

- 894 • The offset is 0
- 895 • The length is 32 bytes (256 bits)

896

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

899 7.2 Examples

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

```
902 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wssu="..."
903     xmlns:xenc="..." xmlns:wsc="..." xmlns:ds="...">
904   <S11:Header>
905     <wsse:Security>
906       <wsc:SecurityContextToken wsu:Id="ctx2">
907         <wsc:Identifier>uuid:...UUID2...</wsc:Identifier>
908       </wsc:SecurityContextToken>
909       <wsc:DerivedKeyToken wsu:Id="dk2">
910         <wsse:SecurityTokenReference>
911           <wsse:Reference URI="#ctx2"/>
912         </wsse:SecurityTokenReference>
913         <wsc:Nonce>KJHFRE...</wsc:Nonce>
914       </wsc:DerivedKeyToken>
915     <xenc:ReferenceList>
916       ...
917     <ds:KeyInfo>
918       <wsse:SecurityTokenReference>
919         <wsse:Reference URI="#dk2"/>
920       </wsse:SecurityTokenReference>
921     </ds:KeyInfo>
922     ...
923   </xenc:ReferenceList>
924   <wsc:SecurityContextToken wsu:Id="ctx1">
925     <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
926   </wsc:SecurityContextToken>
927   <wsc:DerivedKeyToken wsu:Id="dk1">
928     <wsse:SecurityTokenReference>
929       <wsse:Reference URI="#ctx1"/>
930     </wsse:SecurityTokenReference>
931     <wsc:Nonce>KJHFRE...</wsc:Nonce>
932   </wsc:DerivedKeyToken>
933   <xenc:ReferenceList>
934     ...
935   <ds:KeyInfo>
936     <wsse:SecurityTokenReference>
937       <wsse:Reference URI="#dk1"/>
938     </wsse:SecurityTokenReference>
939   </ds:KeyInfo>
940   ...
941   </xenc:ReferenceList>
942 </wsse:Security>
```

```
943     ...
944     </S11:Header>
945     <S11:Body>
946         ...
947     </S11:Body>
948 </S11:Envelope>
```

949

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

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

958

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

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

969

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

976

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

982 7.3 Implied Derived Keys

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

990

991 Consequently, the following two illustrations are functionally equivalent:

```

992     <wsse:Security xmlns:wsc="..." xmlns:wsse="..." xmlns:xx="..."
993 xmlns:ds="..." xmlns:wsu="...">
994     <xx:MyToken wsu:Id="base">...</xx:MyToken>
995     <wsc:DerivedKeyToken wsu:Id="newKey">
996     <wsse:SecurityTokenReference>
997     <wsse:Reference URI="#base"/>
998     </wsse:SecurityTokenReference>
999     <wsc:Nonce>...</wsc:Nonce>
1000 </wsc:DerivedKeyToken>
1001 <ds:Signature>
1002     ...
1003 <ds:KeyInfo>
1004     <wsse:SecurityTokenReference>
1005     <wsse:Reference URI="#newKey"/>
1006     </wsse:SecurityTokenReference>
1007     </ds:KeyInfo>
1008 </ds:Signature>
1009 </wsse:Security>

```

1010

1011 This is functionally equivalent to the following:

```

1012     <wsse:Security xmlns:wsc="..." xmlns:wsse="..." xmlns:xx="..."
1013 xmlns:ds="..." xmlns:wsu="...">
1014     <xx:MyToken wsu:Id="base">...</xx:MyToken>
1015     <ds:Signature>
1016     ...
1017     <ds:KeyInfo>
1018     <wsse:SecurityTokenReference wsc:Nonce="...">
1019     <wsse:Reference URI="#base"/>
1020     </wsse:SecurityTokenReference>
1021     </ds:KeyInfo>
1022 </ds:Signature>
1023 </wsse:Security>

```

8 Associating a Security Context

1024

1025 For a variety of reasons it may be necessary to reference a Security Context Token. These references
1026 can be broken into two general categories: references from within the `<wsse:Security>` element,
1027 generally used to indicate the key used in a signature or encryption operation and references from other
1028 parts of the SOAP envelope, for example to specify a token to be used in some particular way.
1029 References within the `<wsse:Security>` element can further be divided into reference to an SCT
1030 found within the message and references to a SCT not present in the message.

1031

1032 The Security Context Token does not support references to it using key identifiers or key names. All
1033 references **MUST** either use an ID (to a `wsu:Id` attribute) or a `<wsse:Reference>` to the
1034 `<wsc:Identifier>` element.

1035

1036 References using an ID are message-specific. References using the `<wsc:Identifier>` element value
1037 are message independent.

1038

1039 If the SCT is referenced from within the `<wsse:Security>` element or from an RST or RSTR, it is
1040 **RECOMMENDED** that these references be message independent, but these references **MAY** be
1041 message-specific. A reference from the RST/RSTR is treated differently than other references from the
1042 SOAP Body as the RST/RSTR is exclusively dealing with security related information similar to the
1043 `<wsse:Security>` element.

1044

1045 When an SCT located in the `<wsse:Security>` element is referenced from outside the
1046 `<wsse:Security>` element, a message independent referencing mechanisms **MUST** be used, to
1047 enable a cleanly layered processing model unless there is a prior agreement between the involved parties
1048 to use message-specific referencing mechanism.

1049

1050 When an SCT is referenced from within the `<wsse:Security>` element, but the SCT is not present in
1051 the message, (presumably because it was transmitted in a previous message) a message independent
1052 referencing mechanism **MUST** be used.

1053

1054 The following example illustrates associating a specific security context with an action.

1055

```
1056 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
1057   xmlns:wsc="...">  
1058   <S11:Header>  
1059     ...  
1060     <wsse:Security>  
1061       <wsc:SecurityContextToken wsu:Id="sct1">  
1062         <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>  
1063       </wsc:SecurityContextToken>  
1064       <ds:Signature xmlns:ds="...">  
1065         ...signature over body and crucial headers using #sct1...  
1066       </ds:Signature>  
1067       <wsc:SecurityContextToken wsu:Id="sct2">  
1068         <wsc:Identifier>uuid:...UUID2...</wsc:Identifier>  
1069       </wsc:SecurityContextToken>  
1070       <ds:Signature xmlns:ds="...">  
1071         ...signature over body and crucial headers using #sct2...  
1072       </ds:Signature>  
1073     </wsse:Security>
```

1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083

```
    ...  
</S11:Header>  
<S11:Body wsu:Id="req">  
  <xx:Custom xmlns:xx="http://example.com/custom" xmlns:wsse="...">  
    ...  
    <wsse:SecurityTokenReference>  
      <wsse:Reference URI="uuid:...UUID2..." />  
    </wsse:SecurityTokenReference>  
  </xx:Custom>  
</S11:Body>  
</S11:Envelope>
```

1084

9 Error Handling

1085 There are many circumstances where an *error* can occur while processing security information. Errors
1086 use the SOAP Fault mechanism. Note that the reason text provided below is RECOMMENDED, but
1087 alternative text MAY be provided if more descriptive or preferred by the implementation. The tables
1088 below are defined in terms of SOAP 1.1. For SOAP 1.2, the Fault/Code/Value is env:Sender (as defined
1089 in SOAP 1.2) and the Fault/Code/Subcode/Value is the *faultcode* below and the Fault/Reason/Text is the
1090 *faultstring* below. It should be noted that profiles MAY provide second-level details fields, but they should
1091 be careful not to introduce security vulnerabilities when doing so (e.g. by providing too detailed
1092 information).

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

1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118

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 [\[WS-Trust\]](#).

1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141

11 Conformance

An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level requirements defined within this specification. A SOAP Node MUST NOT use the XML namespace identifier for this specification (listed in Section 1.3) within SOAP Envelopes unless it is compliant with this specification.

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

Additionally normative text within this specification takes precedence over normative outlines (as described in section 1.5.1), which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further constrains the schemas and/or WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

Compliant services are NOT REQUIRED to implement everything defined in this specification. However, if a service implements an aspect of the specification, it MUST comply with the requirements specified (e.g. related "MUST" statements). If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for any other unrecognized/unsupported message. If an OPTIONAL message is supported, then the implementation MUST satisfy all of the MUST and REQUIRED sections of the message.

1142

A. Sample Usages

1143 This non-normative appendix illustrates several sample usage patterns of [WS-Trust] and this document.
1144 Specifically, it illustrates different patterns that could be used to parallel, at an end-to-end message level,
1145 the selected TLS/SSL scenarios. This is not intended to be the definitive method for the scenarios, nor is
1146 it fully inclusive. Its purpose is simply to illustrate, in a context familiar to readers, how this specification
1147 might be used.

1148 The following sections are based on a scenario where the client wishes to authenticate the server prior to
1149 sharing any of its own credentials.

1150

1151 It should be noted that the following sample usages are illustrative; any implementation of the examples
1152 illustrated below should be carefully reviewed for potential security attacks. For example, multi-leg
1153 exchanges such as those below should be careful to prevent man-in-the-middle attacks or downgrade
1154 attacks. It may be desirable to use running hashes as challenges that are signed or a similar mechanism
1155 to ensure continuity of the exchange.

1156 The examples below assume that both parties understand the appropriate security policies in use and
1157 can correctly construct signatures and encryption that the other party can process.

A.1 Anonymous SCT

1158 In this scenario the requestor wishes to remain anonymous while authenticating the recipient and
1159 establishing an SCT for secure communication.

1160

1161 This scenario assumes that the requestor has a key for the recipient. If this isn't the case, they can use
1162 [WS-MEX] or the mechanisms described in a later section or obtain one from another security token
1163 service.
1164

1165

1166 There are two basic patterns that can apply, which only vary slightly. The first is as follows:

- 1167 1. The requestor sends an RST to the recipient requesting an SCT. The request contains key
1168 material encrypted for the recipient. The request is not authenticated.
- 1169 2. The recipient, if it accepts such requests, returns an RSTRC with one or more RSTRs with the
1170 SCT as the requested token and does not return any proof information indicating that the
1171 requestor's key is the proof.

1172 A slight variation on this is as follows:

- 1173 1. The requestor sends an RST to the recipient requesting an SCT. The request contains key
1174 material encrypted for the recipient. The request is not authenticated.
- 1175 2. The recipient, if it accepts such requests, returns an RSTRC with one or more RSTR and with the
1176 SCT as the requested token and returns its own key material encrypted using the requestor's key.

1177

1178 Another slight variation is to return a new key encrypted using the requestor's provided key.

1179 It should be noted that the variations that involve encrypting data using the requestor's key material might
1180 be subject to certain types of key attacks.

1181 Yet another approach is to establish a secure channel (e.g. TLS/SSL IP/Sec) between the requestor and
1182 the recipient. Key material can then safely flow in either direction. In some circumstances, this provides
1183 greater protection than the approach above when returning key information to the requestor.

1184 **A.2 Mutual Authentication SCT**

1185 In this scenario the requestor is willing to authenticate, but wants the recipient to authenticate first. The
1186 following steps outline the message flow:

- 1187 1. The requestor sends an RST requesting an SCT. The request contains key material encrypted
1188 for the recipient. The request is not authenticated.
- 1189 2. The recipient returns an RSTRC with one or more RSTRs including a challenge for the requestor.
1190 The RSTRC is secured by the recipient so that the requestor can authenticate it.
- 1191 3. The requestor, after authenticating the recipient's RSTRC, sends an RSTRC responding to the
1192 challenge.
- 1193 4. The recipient, after authenticating the requestor's RSTRC, sends a secured RSTRC containing
1194 the token and either proof information or partial key material (depending on whether or not the
1195 requestor provided key material).

1196

1197 Another variation exists where step 1 includes a specific challenge for the service. Depending on the
1198 type of challenge used this may not be necessary because the message may contain enough entropy to
1199 ensure a fresh response from the recipient.

1200

1201 In other variations the requestor doesn't include key information until step 3 so that it can first verify the
1202 signature of the recipient in step 2.

1203

B. Token Discovery Using RST/RSTR

1204 If the recipient's security token is not known, the RST/RSTR mechanism can still be used. The following
1205 example illustrates one possible sequence of messages:

- 1206 1. The requestor sends an RST requesting an SCT. This request does not contain any key
1207 material, nor is the request authenticated.
- 1208 2. The recipient sends an RSTRC with one or more RSTRs to the requestor with an embedded
1209 challenge. The RSTRC is secured by the recipient so that the requestor can authenticate it.
- 1210 3. The requestor sends an RSTRC to the recipient and includes key information protected for the
1211 recipient. This request may or may not be secured depending on whether or not the request is
1212 anonymous.
- 1213 4. The final issuance step depends on the exact scenario. Any of the final legs from above might be
1214 used.

1215

1216 Note that step 1 might include a challenge for the recipient. Please refer to the comment in the previous
1217 section on this scenario.

1218 Also note that in response to step 1 the recipient might issue a fault secured with [\[WS-Security\]](#) providing
1219 the requestor with information about the recipient's security token.

1220 C. Acknowledgements

1221 The following individuals have participated in the creation of this specification and are gratefully
1222 acknowledged:

1223 **Original Authors of the initial contribution:**

1224 Steve Anderson, OpenNetwork
1225 Jeff Bohren, OpenNetwork
1226 Toufic Boubez, Layer 7
1227 Marc Chanliou, Computer Associates
1228 Giovanni Della-Libera, Microsoft
1229 Brendan Dixon, Microsoft
1230 Praerit Garg, Microsoft
1231 Martin Gudgin (Editor), Microsoft
1232 Satoshi Hada, IBM
1233 Phillip Hallam-Baker, VeriSign
1234 Maryann Hondo, IBM
1235 Chris Kaler, Microsoft
1236 Hal Lockhart, BEA
1237 Robin Martherus, Oblix
1238 Hiroshi Maruyama, IBM
1239 Anthony Nadalin (Editor), IBM
1240 Nataraj Nagaratnam, IBM
1241 Andrew Nash, Reactivity
1242 Rob Philpott, RSA Security
1243 Darren Platt, Ping Identity
1244 Hemma Prafullchandra, VeriSign
1245 Maneesh Sahu, Actional
1246 John Shewchuk, Microsoft
1247 Dan Simon, Microsoft
1248 Davanum Srinivas, Computer Associates
1249 Elliot Waingold, Microsoft
1250 David Waite, Ping Identity
1251 Doug Walter, Microsoft
1252 Riaz Zolfonoon, RSA Security

1253

1254 **Original Acknowledgements of the initial contribution:**

1255 Paula Austel, IBM
1256 Keith Ballinger, Microsoft
1257 John Brezak, Microsoft
1258 Tony Cowan, IBM
1259 HongMei Ge, Microsoft
1260 Slava Kavsan, RSA Security
1261 Scott Konersmann, Microsoft
1262 Leo Laferriere, Computer Associates
1263 Paul Leach, Microsoft
1264 Richard Levinson, Computer Associates
1265 John Linn, RSA Security
1266 Michael McIntosh, IBM
1267 Steve Millet, Microsoft

- 1268 Birgit Pfitzmann, IBM
- 1269 Fumiko Satoh, IBM
- 1270 Keith Stobie, Microsoft
- 1271 T.R. Vishwanath, Microsoft
- 1272 Richard Ward, Microsoft
- 1273 Hervey Wilson, Microsoft
- 1274 **TC Members during the development of this specification:**
- 1275 Don Adams, Tibco Software Inc.
- 1276 Jan Alexander, Microsoft Corporation
- 1277 Steve Anderson, BMC Software
- 1278 Donal Arundel, IONA Technologies
- 1279 Howard Bae, Oracle Corporation
- 1280 Abbie Barbir, Nortel Networks Limited
- 1281 Charlton Barreto, Adobe Systems
- 1282 Mighael Botha, Software AG, Inc.
- 1283 Toufic Boubez, Layer 7 Technologies Inc.
- 1284 Norman Brickman, Mitre Corporation
- 1285 Melissa Brumfield, Booz Allen Hamilton
- 1286 Geoff Bullen, Microsoft Corporation
- 1287 Lloyd Burch, Novell
- 1288 Scott Cantor, Internet2
- 1289 Greg Carpenter, Microsoft Corporation
- 1290 Steve Carter, Novell
- 1291 Ching-Yun (C.Y.) Chao, IBM
- 1292 Martin Chapman, Oracle Corporation
- 1293 Kate Cherry, Lockheed Martin
- 1294 Henry (Hyenvui) Chung, IBM
- 1295 Luc Clement, Systinet Corp.
- 1296 Paul Cotton, Microsoft Corporation
- 1297 Glen Daniels, Sonic Software Corp.
- 1298 Peter Davis, Neustar, Inc.
- 1299 Martijn de Boer, SAP AG
- 1300 Duane DeCouteau, Veterans Health Administration
- 1301 Werner Dittmann, Siemens AG
- 1302 Abdeslem DJAOUI, CCLRC-Rutherford Appleton Laboratory
- 1303 Fred Dushin, IONA Technologies
- 1304 Petr Dvorak, Systinet Corp.
- 1305 Colleen Evans, Microsoft Corporation
- 1306 Ruchith Fernando, WSO2
- 1307 Mark Fussell, Microsoft Corporation
- 1308 Vijay Gajjala, Microsoft Corporation
- 1309 Marc Goodner, Microsoft Corporation
- 1310 Hans Granqvist, VeriSign
- 1311 Martin Gudgin, Microsoft Corporation

1312 Tony Gullotta, SOA Software Inc.
1313 Jiandong Guo, Sun Microsystems
1314 Phillip Hallam-Baker, VeriSign
1315 Patrick Harding, Ping Identity Corporation
1316 Heather Hinton, IBM
1317 Frederick Hirsch, Nokia Corporation
1318 Jeff Hodges, Neustar, Inc.
1319 Will Hopkins, Oracle Corporation
1320 Alex Hristov, Otecia Incorporated
1321 John Hughes, PA Consulting
1322 Diane Jordan, IBM
1323 Venugopal K, Sun Microsystems
1324 Chris Kaler, Microsoft Corporation
1325 Dana Kaufman, Forum Systems, Inc.
1326 Paul Knight, Nortel Networks Limited
1327 Ramanathan Krishnamurthy, IONA Technologies
1328 Christopher Kurt, Microsoft Corporation
1329 Kelvin Lawrence, IBM
1330 Hubert Le Van Gong, Sun Microsystems
1331 Jong Lee, Oracle Corporation
1332 Rich Levinson, Oracle Corporation
1333 Tommy Lindberg, Dajeil Ltd.
1334 Mark Little, JBoss Inc.
1335 Hal Lockhart, Oracle Corporation
1336 Mike Lyons, Layer 7 Technologies Inc.
1337 Eve Maler, Sun Microsystems
1338 Ashok Malhotra, Oracle Corporation
1339 Anand Mani, CrimsonLogic Pte Ltd
1340 Jonathan Marsh, Microsoft Corporation
1341 Robin Martherus, Oracle Corporation
1342 Miko Matsumura, Infravio, Inc.
1343 Gary McAfee, IBM
1344 Michael McIntosh, IBM
1345 John Merrells, Sxip Networks SRL
1346 Jeff Mischkinisky, Oracle Corporation
1347 Prateek Mishra, Oracle Corporation
1348 Bob Morgan, Internet2
1349 Vamsi Motukuru, Oracle Corporation
1350 Raajmohan Na, EDS
1351 Anthony Nadalin, IBM
1352 Andrew Nash, Reactivity, Inc.
1353 Eric Newcomer, IONA Technologies

1354 Duane Nickull, Adobe Systems
1355 Toshihiro Nishimura, Fujitsu Limited
1356 Rob Philpott, RSA Security
1357 Denis Pilipchuk, Oracle Corporation
1358 Darren Platt, Ping Identity Corporation
1359 Martin Raepple, SAP AG
1360 Nick Ragouzis, Enosis Group LLC
1361 Prakash Reddy, CA
1362 Alain Regnier, Ricoh Company, Ltd.
1363 Irving Reid, Hewlett-Packard
1364 Bruce Rich, IBM
1365 Tom Rutt, Fujitsu Limited
1366 Maneesh Sahu, Actional Corporation
1367 Frank Siebenlist, Argonne National Laboratory
1368 Joe Smith, Apani Networks
1369 Davanum Srinivas, WSO2
1370 David Staggs, Veterans Health Administration
1371 Yakov Sverdlov, CA
1372 Gene Thurston, AmberPoint
1373 Victor Valle, IBM
1374 Asir Vedamuthu, Microsoft Corporation
1375 Greg Whitehead, Hewlett-Packard
1376 Ron Williams, IBM
1377 Corinna Witt, Oracle Corporation
1378 Kyle Young, Microsoft Corporation
1379