



# WS-SecureConversation 1.4

## OASIS Committee Draft 02

29 October 2008

### Specification URIs:

#### This 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>

#### Previous Version:

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-01.doc> (Authoritative)  
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-01.pdf>  
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-spec-cd-01.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    | <a href="http://schemas.xmlsoap.org/soap/envelope/">http://schemas.xmlsoap.org/soap/envelope/</a>   | [SOAP]             |
| S12    | <a href="http://www.w3.org/2003/05/soap-envelope">http://www.w3.org/2003/05/soap-envelope</a>   | [SOAP12]           |
| wsu    | <a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd</a> | [WS-Security]      |
| wsse   | <a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd</a>   | [WS-Security]      |
| wst    | <a href="http://docs.oasis-open.org/ws-sx/ws-trust/200512">http://docs.oasis-open.org/ws-sx/ws-trust/200512</a>   | [WS-Trust]         |
| wsc    | <a href="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512">http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512</a>   | This specification |
| wsa    | <a href="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing</a>   | [WS-Addressing]    |
| ds     | <a href="http://www.w3.org/2000/09/xmldsig#">http://www.w3.org/2000/09/xmldsig#</a>   | [XML-Signature]    |
| xenc   | <a href="http://www.w3.org/2001/04/xmlenc#">http://www.w3.org/2001/04/xmlenc#</a>   | [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-xmlenc-core-20021210/>  
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 Standard, "WS-SecurityPolicy 1.3", 2008  
170                               <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802/>  
171

---

## 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-  
227 SecurityPolicy]. 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      (020)      <tru:StockSymbol
267                  xmlns:tru="http://fabrikam123.com/payloads">
268                  QQQ
269                  </tru:StockSymbol>
270 (021)      </S11:Body>
271 (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 MSUT 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 certificate 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="...">
        ...
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  ...
</S11:Envelope>
```

```

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 crucial 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  
570

```
<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">
```

```
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 this specialized binding of the renewal mechanism defined in WS-  
602 Trust. 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 crucial 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  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657

```
        http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew
    </wst:RequestType>
    <wst:RenewTarget>
        <wsse:SecurityTokenReference>
            <wsse:Reference URI="uuid:...UUID1..." />
        </wsse:SecurityTokenReference>
    </wst:RenewTarget>
    <wst:Lifetime>...</wst:Lifetime>
</wst:RequestSecurityToken>
</S11:Body>
</S11:Envelope>
```

658

659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680

```
<S11:Envelope xmlns:S11="..." xmlns:wst="..." xmlns:wsc="...">
  <S11:Header>
    ...
    <wsa:Action xmlns:wsa="...">
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Renew
    </wsa:Action>
    ...
  </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponseCollection>
      <wst:RequestSecurityTokenResponse>
        <wst:RequestedSecurityToken>
          <wsc:SecurityContextToken>
            <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
            <wsc:Instance>UUID2</wsc:Instance>
          </wsc:SecurityContextToken>
        </wst:RequestedSecurityToken>
        <wst:Lifetime>...</wst:Lifetime>
      </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
  </S11:Body>
</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 crucial 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 if 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="uuid:...UUID1..." />  
        </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

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

771

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

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  
837 predefines the `P_SHA1` algorithm described above. If this attribute isn't specified, this algorithm is  
838 assumed.

839 /wsc:DerivedKeyToken/wsse:SecurityTokenReference

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

844 /wsc:DerivedKeyToken/wsc:Properties

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

850 /wsc:DerivedKeyToken/wsc:Properties/wsc:Name

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

853 /wsc:DerivedKeyToken/wsc:Properties/wsc:Label

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

856 /wsc:DerivedKeyToken/wsc:Properties/wsc:Nonce

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

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

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

861 /wsc:DerivedKeyToken/wsc:Generation

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

868 /wsc:DerivedKeyToken/wsc:Offset

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

876 /wsc:DerivedKeyToken/wsc:Length

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

881 /wsc:DerivedKeyToken/wsc:Label

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

885 /wsc:DerivedKeyToken/wsc:Nonce

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

890

891 If additional information is not specified as explicit elements, then the following defaults 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:wsp="..."  
901   xmlns:xenc="..." xmlns:wsc="..." 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,  
1025 generally used to indicate the key used in a signature or encryption operation and references from other  
1026 parts of the SOAP envelope, for example to specify a token to be used in some particular way.  
1027 References within the `<wsse:Security>` element can further be divided into reference to an SCT  
1028 found within the message and references to a SCT not present in the message.

1029

1030 The Security Context Token does not support references to it using key identifiers or key names. All  
1031 references **MUST** either use an ID (to a `wsu:Id` attribute) or a `<wsse:Reference>` to the  
1032 `<wsc:Identifier>` element.

1033

1034 References using an ID are message-specific. References using the `<wsc:Identifier>` element value  
1035 are message independent.

1036

1037 If the SCT is referenced from within the `<wsse:Security>` element or from an RST or RSTR, it is  
1038 **RECOMMENDED** that these references be message independent, but these references **MAY** be  
1039 message-specific. A reference from the RST/RSTR is treated differently than other references from the  
1040 SOAP Body as the RST/RSTR is exclusively dealing with security related information similar to the  
1041 `<wsse:Security>` element.

1042

1043 When an SCT located in the `<wsse:Security>` element is referenced from outside the  
1044 `<wsse:Security>` element, a message independent referencing mechanisms **MUST** be used, to  
1045 enable a cleanly layered processing model unless there is a prior agreement between the involved parties  
1046 to use message-specific referencing mechanism.

1047

1048 When an SCT is referenced from within the `<wsse:Security>` element, but the SCT is not present in  
1049 the message, (presumably because it was transmitted in a previous message) a message independent  
1050 referencing mechanism **MUST** be used.

1051

1052 The following example illustrates associating a specific security context with an action.

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 crucial 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 crucial headers using #sct2...
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
</S11:Envelope>
```

1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081

```
    ...  
</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>
```

1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090

---

## 9 Error Handling

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).

| 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           |



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  
1116

---

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

---

## 1117 **11 Conformance**

1118 An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level  
1119 requirements defined within this specification. A SOAP Node MUST NOT use the XML namespace  
1120 identifier for this specification (listed in Section 1.3) within SOAP Envelopes unless it is compliant with this  
1121 specification.

1122 This specification references a number of other specifications (see the table above). In order to comply  
1123 with this specification, an implementation MUST implement the portions of referenced specifications  
1124 necessary to comply with the required provisions of this specification. Additionally, the implementation of  
1125 the portions of the referenced specifications that are specifically cited in this specification MUST comply  
1126 with the rules for those portions as established in the referenced specification.

1127 Additionally normative text within this specification takes precedence over normative outlines (as  
1128 described in section 1.5.1), which in turn take precedence over the XML Schema [XML Schema Part 1,  
1129 Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further  
1130 constrains the schemas and/or WSDL that are part of this specification; and this specification contains  
1131 further constraints on the elements defined in referenced schemas.

1132 Compliant services are NOT REQUIRED to implement everything defined in this specification. However,  
1133 if a service implements an aspect of the specification, it MUST comply with the requirements specified  
1134 (e.g. related "MUST" statements). If an OPTIONAL message is not supported, then the implementation  
1135 SHOULD Fault just as it would for any other unrecognized/unsupported message. If an OPTIONAL  
1136 message is supported, then the implementation MUST satisfy all of the MUST and REQUIRED sections  
1137 of the message.

1138

1139

1140

## A. Sample Usages

1141 This non-normative appendix illustrates several sample usage patterns of [WS-Trust] and this document.  
1142 Specifically, it illustrates different patterns that could be used to parallel, at an end-to-end message level,  
1143 the selected TLS/SSL scenarios. This is not intended to be the definitive method for the scenarios, nor is  
1144 it fully inclusive. Its purpose is simply to illustrate, in a context familiar to readers, how this specification  
1145 might be used.

1146 The following sections are based on a scenario where the client wishes to authenticate the server prior to  
1147 sharing any of its own credentials.

1148

1149 It should be noted that the following sample usages are illustrative; any implementation of the examples  
1150 illustrated below should be carefully reviewed for potential security attacks. For example, multi-leg  
1151 exchanges such as those below should be careful to prevent man-in-the-middle attacks or downgrade  
1152 attacks. It may be desirable to use running hashes as challenges that are signed or a similar mechanism  
1153 to ensure continuity of the exchange.

1154 The examples below assume that both parties understand the appropriate security policies in use and  
1155 can correctly construct signatures and encryption that the other party can process.

### A.1 Anonymous SCT

1156 In this scenario the requestor wishes to remain anonymous while authenticating the recipient and  
1157 establishing an SCT for secure communication.

1158

1159 This scenario assumes that the requestor has a key for the recipient. If this isn't the case, they can use  
1160 [WS-MEX] or the mechanisms described in a later section or obtain one from another security token  
1161 service.  
1162

1163

1164 There are two basic patterns that can apply, which only vary slightly. The first is as follows:

- 1165 1. The requestor sends an RST to the recipient requesting an SCT. The request contains key  
1166 material encrypted for the recipient. The request is not authenticated.
- 1167 2. The recipient, if it accepts such requests, returns an RSTRC with one or more RSTRs with the  
1168 SCT as the requested token and does not return any proof information indicating that the  
1169 requestor's key is the proof.

1170 A slight variation on this is as follows:

- 1171 1. The requestor sends an RST to the recipient requesting an SCT. The request contains key  
1172 material encrypted for the recipient. The request is not authenticated.
- 1173 2. The recipient, if it accepts such requests, returns an RSTRC with one or more RSTR and with the  
1174 SCT as the requested token and returns its own key material encrypted using the requestor's key.

1175

1176 Another slight variation is to return a new key encrypted using the requestor's provided key.

1177 It should be noted that the variations that involve encrypting data using the requestor's key material might  
1178 be subject to certain types of key attacks.

1179 Yet another approach is to establish a secure channel (e.g. TLS/SSL IP/Sec) between the requestor and  
1180 the recipient. Key material can then safely flow in either direction. In some circumstances, this provides  
1181 greater protection than the approach above when returning key information to the requestor.

## 1182 **A.2 Mutual Authentication SCT**

1183 In this scenario the requestor is willing to authenticate, but wants the recipient to authenticate first. The  
1184 following steps outline the message flow:

- 1185 1. The requestor sends an RST requesting an SCT. The request contains key material encrypted  
1186 for the recipient. The request is not authenticated.
- 1187 2. The recipient returns an RSTRC with one or more RSTRs including a challenge for the requestor.  
1188 The RSTRC is secured by the recipient so that the requestor can authenticate it.
- 1189 3. The requestor, after authenticating the recipient's RSTRC, sends an RSTRC responding to the  
1190 challenge.
- 1191 4. The recipient, after authenticating the requestor's RSTRC, sends a secured RSTRC containing  
1192 the token and either proof information or partial key material (depending on whether or not the  
1193 requestor provided key material).

1194

1195 Another variation exists where step 1 includes a specific challenge for the service. Depending on the  
1196 type of challenge used this may not be necessary because the message may contain enough entropy to  
1197 ensure a fresh response from the recipient.

1198

1199 In other variations the requestor doesn't include key information until step 3 so that it can first verify the  
1200 signature of the recipient in step 2.

---

1201 **B. Token Discovery Using RST/RSTR**

1202 If the recipient's security token is not known, the RST/RSTR mechanism can still be used. The following  
1203 example illustrates one possible sequence of messages:

- 1204       1. The requestor sends an RST requesting an SCT. This request does not contain any key  
1205       material, nor is the request authenticated.
- 1206       2. The recipient sends an RSTRC with one or more RSTRs to the requestor with an embedded  
1207       challenge. The RSTRC is secured by the recipient so that the requestor can authenticate it.
- 1208       3. The requestor sends an RSTRC to the recipient and includes key information protected for the  
1209       recipient. This request may or may not be secured depending on whether or not the request is  
1210       anonymous.
- 1211       4. The final issuance step depends on the exact scenario. Any of the final legs from above might be  
1212       used.

1213

1214 Note that step 1 might include a challenge for the recipient. Please refer to the comment in the previous  
1215 section on this scenario.

1216 Also note that in response to step 1 the recipient might issue a fault secured with [\[WS-Security\]](#) providing  
1217 the requestor with information about the recipient's security token.

1218

---

## C. Acknowledgements

1219 The following individuals have participated in the creation of this specification and are gratefully  
1220 acknowledged:

1221 **Original Authors of the initial contribution:**

1222 Steve Anderson, OpenNetwork  
1223 Jeff Bohren, OpenNetwork  
1224 Toufic Boubez, Layer 7  
1225 Marc Chanliau, Computer Associates  
1226 Giovanni Della-Libera, Microsoft  
1227 Brendan Dixon, Microsoft  
1228 Praerit Garg, Microsoft  
1229 Martin Gudgin (Editor), Microsoft  
1230 Satoshi Hada, IBM  
1231 Phillip Hallam-Baker, VeriSign  
1232 Maryann Hondo, IBM  
1233 Chris Kaler, Microsoft  
1234 Hal Lockhart, BEA  
1235 Robin Martherus, Oblix  
1236 Hiroshi Maruyama, IBM  
1237 Anthony Nadalin (Editor), IBM  
1238 Nataraj Nagaratnam, IBM  
1239 Andrew Nash, Reactivity  
1240 Rob Philpott, RSA Security  
1241 Darren Platt, Ping Identity  
1242 Hemma Prafullchandra, VeriSign  
1243 Maneesh Sahu, Actional  
1244 John Shewchuk, Microsoft  
1245 Dan Simon, Microsoft  
1246 Davanum Srinivas, Computer Associates  
1247 Elliot Waingold, Microsoft  
1248 David Waite, Ping Identity  
1249 Doug Walter, Microsoft  
1250 Riaz Zolfonoon, RSA Security

1251

1252 **Original Acknowledgements of the initial contribution:**

1253 Paula Austel, IBM  
1254 Keith Ballinger, Microsoft  
1255 John Brezak, Microsoft  
1256 Tony Cowan, IBM  
1257 HongMei Ge, Microsoft  
1258 Slava Kavsan, RSA Security  
1259 Scott Konersmann, Microsoft  
1260 Leo Laferriere, Computer Associates  
1261 Paul Leach, Microsoft  
1262 Richard Levinson, Computer Associates  
1263 John Linn, RSA Security  
1264 Michael McIntosh, IBM  
1265 Steve Millet, Microsoft

- 1266 Birgit Pfitzmann, IBM  
1267 Fumiko Satoh, IBM  
1268 Keith Stobie, Microsoft  
1269 T.R. Vishwanath, Microsoft  
1270 Richard Ward, Microsoft  
1271 Hervey Wilson, Microsoft  
1272 **TC Members during the development of this specification:**  
1273 Don Adams, Tibco Software Inc.  
1274 Jan Alexander, Microsoft Corporation  
1275 Steve Anderson, BMC Software  
1276 Donal Arundel, IONA Technologies  
1277 Howard Bae, Oracle Corporation  
1278 Abbie Barbir, Nortel Networks Limited  
1279 Charlton Barreto, Adobe Systems  
1280 Mighael Botha, Software AG, Inc.  
1281 Toufic Boubetz, Layer 7 Technologies Inc.  
1282 Norman Brickman, Mitre Corporation  
1283 Melissa Brumfield, Booz Allen Hamilton  
1284 Geoff Bullen, Microsoft Corporation  
1285 Lloyd Burch, Novell  
1286 Scott Cantor, Internet2  
1287 Greg Carpenter, Microsoft Corporation  
1288 Steve Carter, Novell  
1289 Ching-Yun (C.Y.) Chao, IBM  
1290 Martin Chapman, Oracle Corporation  
1291 Kate Cherry, Lockheed Martin  
1292 Henry (Hyenvui) Chung, IBM  
1293 Luc Clement, Systinet Corp.  
1294 Paul Cotton, Microsoft Corporation  
1295 Glen Daniels, Sonic Software Corp.  
1296 Peter Davis, Neustar, Inc.  
1297 Martijn de Boer, SAP AG  
1298 Duane DeCouteau, Veterans Health Administration  
1299 Werner Dittmann, Siemens AG  
1300 Abdeslem DJAOUI, CCLRC-Rutherford Appleton Laboratory  
1301 Fred Dushin, IONA Technologies  
1302 Petr Dvorak, Systinet Corp.  
1303 Colleen Evans, Microsoft Corporation  
1304 Ruchith Fernando, WSO2  
1305 Mark Fussell, Microsoft Corporation  
1306 Vijay Gajjala, Microsoft Corporation  
1307 Marc Goodner, Microsoft Corporation  
1308 Hans Granqvist, VeriSign  
1309 Martin Gudgin, Microsoft Corporation

1310 Tony Gullotta, SOA Software Inc.  
1311 Jiandong Guo, Sun Microsystems  
1312 Phillip Hallam-Baker, VeriSign  
1313 Patrick Harding, Ping Identity Corporation  
1314 Heather Hinton, IBM  
1315 Frederick Hirsch, Nokia Corporation  
1316 Jeff Hodges, Neustar, Inc.  
1317 Will Hopkins, BEA Systems, Inc.  
1318 Alex Hristov, Otecia Incorporated  
1319 John Hughes, PA Consulting  
1320 Diane Jordan, IBM  
1321 Venugopal K, Sun Microsystems  
1322 Chris Kaler, Microsoft Corporation  
1323 Dana Kaufman, Forum Systems, Inc.  
1324 Paul Knight, Nortel Networks Limited  
1325 Ramanathan Krishnamurthy, IONA Technologies  
1326 Christopher Kurt, Microsoft Corporation  
1327 Kelvin Lawrence, IBM  
1328 Hubert Le Van Gong, Sun Microsystems  
1329 Jong Lee, BEA Systems, Inc.  
1330 Rich Levinson, Oracle Corporation  
1331 Tommy Lindberg, Dajeil Ltd.  
1332 Mark Little, JBoss Inc.  
1333 Hal Lockhart, BEA Systems, Inc.  
1334 Mike Lyons, Layer 7 Technologies Inc.  
1335 Eve Maler, Sun Microsystems  
1336 Ashok Malhotra, Oracle Corporation  
1337 Anand Mani, CrimsonLogic Pte Ltd  
1338 Jonathan Marsh, Microsoft Corporation  
1339 Robin Martherus, Oracle Corporation  
1340 Miko Matsumura, Infravio, Inc.  
1341 Gary McAfee, IBM  
1342 Michael McIntosh, IBM  
1343 John Merrells, Sxip Networks SRL  
1344 Jeff Mischkinisky, Oracle Corporation  
1345 Prateek Mishra, Oracle Corporation  
1346 Bob Morgan, Internet2  
1347 Vamsi Motukuru, Oracle Corporation  
1348 Raajmohan Na, EDS  
1349 Anthony Nadalin, IBM  
1350 Andrew Nash, Reactivity, Inc.  
1351 Eric Newcomer, IONA Technologies



1352 Duane Nickull, Adobe Systems  
1353 Toshihiro Nishimura, Fujitsu Limited  
1354 Rob Philpott, RSA Security  
1355 Denis Pilipchuk, BEA Systems, Inc.  
1356 Darren Platt, Ping Identity Corporation  
1357 Martin Raepple, SAP AG  
1358 Nick Ragouzis, Enosis Group LLC  
1359 Prakash Reddy, CA  
1360 Alain Regnier, Ricoh Company, Ltd.  
1361 Irving Reid, Hewlett-Packard  
1362 Bruce Rich, IBM  
1363 Tom Rutt, Fujitsu Limited  
1364 Maneesh Sahu, Actional Corporation  
1365 Frank Siebenlist, Argonne National Laboratory  
1366 Joe Smith, Apani Networks  
1367 Davanum Srinivas, WSO2  
1368 David Staggs, Veterans Health Administration  
1369 Yakov Sverdlov, CA  
1370 Gene Thurston, AmberPoint  
1371 Victor Valle, IBM  
1372 Asir Vedamuthu, Microsoft Corporation  
1373 Greg Whitehead, Hewlett-Packard  
1374 Ron Williams, IBM  
1375 Corinna Witt, BEA Systems, Inc.  
1376 Kyle Young, Microsoft Corporation  
1377