



# WS-SecureConversation 1.4

## OASIS Committee Draft 01

9 July 2008

### Location:

This Version:

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-cd-01.doc> (Authoritative)

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-cd-01.pdf>

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/cd/ws-secureconversation-1.4-cd-01.html>

Previous Version:

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.doc>

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.pdf>

<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.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. OASIS trademark, IPR and other policies apply.

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]

## 1.4 Schema File

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

```
http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3.xsd
```

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

## 1.5 Terminology

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

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

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

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

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

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

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

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

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

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

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

## 1.5.1 Notational Conventions

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

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

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

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

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

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

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

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

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

## 1.6 Normative References

- |                        |   |
|------------------------|---|
| <b>[RFC2119]</b>       | S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.<br><a href="http://www.ietf.org/rfc/rfc2119.txt">http://www.ietf.org/rfc/rfc2119.txt</a>  |
| <b>[RFC2246]</b>       | IETF Standard, "The TLS Protocol", January 1999.<br><a href="http://www.ietf.org/rfc/rfc2246.txt">http://www.ietf.org/rfc/rfc2246.txt</a>   |
| <b>[SOAP]</b>          | W3C Note, "SOAP: Simple Object Access Protocol 1.1", 08 May 2000.<br><a href="http://www.w3.org/TR/2000/NOTE-SOAP-20000508/">http://www.w3.org/TR/2000/NOTE-SOAP-20000508/</a>  |
| <b>[SOAP12]</b>        | W3C Recommendation, "SOAP 1.2 Part 1: Messaging Framework", 24 June 2003.<br><a href="http://www.w3.org/TR/2003/REC-soap12-part1-20030624/">http://www.w3.org/TR/2003/REC-soap12-part1-20030624/</a>  |
| <b>[URI]</b>           | T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 3986, MIT/LCS, Day Software, Adobe Systems, January 2005.<br><a href="http://www.ietf.org/rfc/rfc3986.txt">http://www.ietf.org/rfc/rfc3986.txt</a>  |
| <b>[WS-Addressing]</b> | W3C Recommendation, "Web Services Addressing (WS-Addressing)", 9 May 2006.<br><a href="http://www.w3.org/TR/2006/REC-ws-addr-core-20060509">http://www.w3.org/TR/2006/REC-ws-addr-core-20060509</a>   |
| <b>[WS-Security]</b>   | OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", March 2004.<br><a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf</a><br>OASIS Standard, "OASIS Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", February 2006.<br><a href="http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf">http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf</a> |
| <b>[WS-Trust]</b>      | OASIS Standard, "WS-Trust 1.4", 2008<br><a href="http://docs.oasis-open.org/ws-sx/ws-trust/200802">http://docs.oasis-open.org/ws-sx/ws-trust/200802</a>   |



- 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/>

## 1.7 Non-Normative References

- 162       **[WS-MEX]**               "Web Services Metadata Exchange (WS-MetadataExchange)", BEA, Computer  
163                               Associates, IBM, Microsoft, SAP, Sun Microsystems, Inc., webMethods,  
164                               September 2004.  
165       **[WS-SecurityPolicy]**   OASIS Standard, "WS-SecurityPolicy 1.3", 2008  
166                               <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802>  
167

---

## 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}`

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

212 /wsc:SecurityContextToken/{any}

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

214  
215 The <wsc:SecurityContextToken> token elements MUST be preserved. That is, whatever elements  
216 contained within the tag on creation MUST be preserved wherever the token is used. A consumer of a  
217 <wsc:SecurityContextToken> token MAY extend the token by appending information.  
218 Consequently producers of <wsc:SecurityContextToken> tokens should consider this fact when  
219 processing previously generated tokens. A service consuming (processing) a  
220 <wsc:SecurityContextToken> token MAY fault if it discovers an element or attribute inside the token  
221 that it doesn't understand, or it MAY ignore it. The fault code wsc:UnsupportedContextToken is  
222 RECOMMENDED if a fault is raised. The behavior is specified by the services policy [WS-  
223 SecurityPolicy]. Care should be taken when adding information to tokens to ensure that relying parties  
224 can ensure the information has not been altered since the SCT definition does not require a specific way  
225 to secure its contents (which as noted above can be appended to).

226  
227 Security contexts, like all security tokens, can be referenced using the mechanisms described in [WS-  
228 Security] (the <wsse:SecurityTokenReference> element referencing the wsu:Id attribute relative to  
229 the XML base document or referencing using the <wsc:Identifier> element's absolute URI). When a  
230 token is referenced, the associated key is used. If a token provides multiple keys then specific bindings  
231 and profiles MUST describe how to reference the separate keys. If a specific key instance needs to be  
232 referenced, then the global attribute wsc:Instance is included in the <wsse:Reference> sub-element  
233 (only when using <wsc:Identifier> references) of the <wsse:SecurityTokenReference>  
234 element as illustrated below:

```
235 <wsse:SecurityTokenReference xmlns:wsse="..." xmlns:wsc="...">  
236 <wsse:Reference URI="uuid:..." wsc:Instance="..."/>  
237 </wsse:SecurityTokenReference>
```

238  
239 The following sample message illustrates the use of a security context token. In this example a context  
240 has been established and the secret is known to both parties. This secret is used to sign the message  
241 body.

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

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

268

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

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

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

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

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

279

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

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

---

### 3 Establishing Security Contexts

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

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

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

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

If an SCT is received, but the key sizes are not supported, then a fault SHOULD be generated using the `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>
```

```

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

```

```

400 <S11:Envelope xmlns:S11="..."
401     xmlns:wst="..." xmlns:wsc="..." xmlns:xenc="...">
402     <S11:Header>
403         ...
404         <wsa:Action xmlns:wsa="...">
405             http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT
406         </wsa:Action>
407         ...
408     </S11:Header>
409     <S11:Body>
410         <wst:RequestSecurityTokenResponseCollection>
411             <wst:RequestSecurityTokenResponse>
412                 <wst:RequestedSecurityToken>
413                     <wsc:SecurityContextToken>
414                         <wsc:Identifier>uuid:...</wsc:Identifier>
415                     </wsc:SecurityContextToken>
416                 </wst:RequestedSecurityToken>
417                 <wst:RequestedProofToken>
418                     <xenc:EncryptedKey Id="newProof">
419                         ...
420                     </xenc:EncryptedKey>
421                 </wst:RequestedProofToken>
422             </wst:RequestSecurityTokenResponse>
423         </wst:RequestSecurityTokenResponseCollection>
424     </S11:Body>
425 </S11:Envelope>

```

### 3.3 SCT Request Example with Target Scope

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

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

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

```

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

```

```

435     xmlns:wst="..." xmlns:xenc="..." xmlns:wsp="..." xmlns:wsa="...">
436 <S11:Header>
437     ...
438     <wsa:Action xmlns:wsa="...">
439         http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT
440     </wsa:Action>
441     ...
442     <wsse:Security>
443         ...
444     </wsse:Security>
445     ...
446 </S11:Header>
447 <S11:Body wsu:Id="req">
448     <wst:RequestSecurityToken>
449         <wst:RequestType>
450             http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
451         </wst:RequestType>
452         <wsp:AppliesTo>
453             <wsa:EndpointReference>
454                 <wsa:Address>http://example.org/webService</wsa:Address>
455             </wsa:EndpointReference>
456         </wsp:AppliesTo>
457     </wst:RequestSecurityToken>
458 </S11:Body>
459 </S11:Envelope>

```

```

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

```



### 3.4 SCT Propagation Example

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

```
<S11:Envelope xmlns:S11="..."
  xmlns:wst="..." xmlns:wsc="..." xmlns:xenc="..." >
  <S11:Header>
    ...
  </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponse>
      <wst:RequestedSecurityToken>
        <wsc:SecurityContextToken>
          <wsc:Identifier>uuid:...</wsc:Identifier>
        </wsc:SecurityContextToken>
      </wst:RequestedSecurityToken>
      <wst:RequestedProofToken>
        <xenc:EncryptedKey Id="newProof">
          ...
        </xenc:EncryptedKey>
      </wst:RequestedProofToken>
    </wst:RequestSecurityTokenResponse>
  </S11:Body>
</S11:Envelope>
```

---

## 4 Amending Contexts

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

The following Action URIs are used with this binding:

```
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
```

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

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

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

This binding uses the request type from the issuance binding.

```
<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:Signature>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#sct"/>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      ...
    </ds:Signature>
  </wsse:Security>
  ...
</S11:Header>
<S11:Body wsu:Id="req">
```

```

567     <wst:RequestSecurityToken>
568         <wst:RequestType>
569             http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
570         </wst:RequestType>
571     </wst:RequestSecurityToken>
572 </S11:Body>
573 </S11:Envelope>

```

574

```

575 <S11:Envelope xmlns:S11="..." xmlns:wst="..." xmlns:wsc="...">
576     <S11:Header>
577         ...
578         <wsa:Action xmlns:wsa="...">
579             http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Amend
580         </wsa:Action>
581         ...
582     </S11:Header>
583     <S11:Body>
584         <wst:RequestSecurityTokenResponseCollection>
585             <wst:RequestSecurityTokenResponse>
586                 <wst:RequestedSecurityToken>
587                     <wsc:SecurityContextToken>
588                         <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
589                     </wsc:SecurityContextToken>
590                 </wst:RequestedSecurityToken>
591             </wst:RequestSecurityTokenResponse>
592         </wst:RequestSecurityTokenResponseCollection>
593     </S11:Body>
594 </S11:Envelope>

```

---

## 5 Renewing Contexts

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

```
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
```

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

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

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

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

This binding uses the request type from the renewal binding.

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

```
<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>
```

```

643         http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew
644     </wst:RequestType>
645     <wst:RenewTarget>
646         <wsse:SecurityTokenReference>
647             <wsse:Reference URI="uuid:...UUID1..."/>
648         </wsse:SecurityTokenReference>
649     </wst:RenewTarget>
650     <wst:Lifetime>...</wst:Lifetime>
651 </wst:RequestSecurityToken>
652 </S11:Body>
653 </S11:Envelope>

```

654

```

655 <S11:Envelope xmlns:S11="..." xmlns:wst="..." xmlns:wsc="...">
656     <S11:Header>
657         ...
658         <wsa:Action xmlns:wsa="...">
659             http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Renew
660         </wsa:Action>
661         ...
662     </S11:Header>
663     <S11:Body>
664         <wst:RequestSecurityTokenResponseCollection>
665             <wst:RequestSecurityTokenResponse>
666                 <wst:RequestedSecurityToken>
667                     <wsc:SecurityContextToken>
668                         <wsc:Identifier>uuid:...UUID1...</wsc:Identifier>
669                         <wsc:Instance>UUID2</wsc:Instance>
670                     </wsc:SecurityContextToken>
671                 </wst:RequestedSecurityToken>
672                 <wst:Lifetime>...</wst:Lifetime>
673             </wst:RequestSecurityTokenResponse>
674         </wst:RequestSecurityTokenResponseCollection>
675     </S11:Body>
676 </S11:Envelope>

```

---

## 6 Canceling Contexts

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

The following Action URIs are used with this binding:

```
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
```

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

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

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

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

The following example illustrates canceling a context.

```
<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>
```

727	</S11:Body>
728	</S11:Envelope>
729	
730	<S11:Envelope xmlns:S11="..." xmlns:wst="..." >
731	<S11:Header>
732	...
733	<wsa:Action xmlns:wsa="...">
734	http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Cancel
735	</wsa:Action>
736	...
737	</S11:Header>
738	<S11:Body>
739	<wst:RequestSecurityTokenResponseCollection>
740	<wst:RequestSecurityTokenResponse>
741	<wst:RequestedTokenCancelled/>
742	</wst:RequestSecurityTokenResponse>
743	</wst:RequestSecurityTokenResponseCollection>
744	</S11:Body>
745	</S11:Envelope>

---

## 7 Deriving Keys

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

## 7.1 Syntax

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

```
<wsc:DerivedKeyToken wsu:Id="..." Algorithm="..." xmlns:wsc="..."
xmlns:wsse="..." xmlns:wsu="...">
  <wsse:SecurityTokenReference>...</wsse:SecurityTokenReference>
  <wsc:Properties>...</wsc:Properties>
  <wsc:Generation>...</wsc:Generation>
  <wsc:Offset>...</wsc:Offset>
  <wsc:Length>...</wsc:Length>
  <wsc:Label>...</wsc:Label>
  <wsc:Nonce>...</wsc:Nonce>
</wsc:DerivedKeyToken>
```

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

`/wsc:DerivedKeyToken`

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

`/wsc:DerivedKeyToken/@wsu:Id`

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

`/wsc:DerivedKeyToken/@Algorithm`

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

835 /wsc:DerivedKeyToken/wsse:SecurityTokenReference

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

840 /wsc:DerivedKeyToken/wsc:Properties

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

846 /wsc:DerivedKeyToken/wsc:Properties/wsc:Name

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

849 /wsc:DerivedKeyToken/wsc:Properties/wsc:Label

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

852 /wsc:DerivedKeyToken/wsc:Properties/wsc:Nonce

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

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

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

857 /wsc:DerivedKeyToken/wsc:Generation

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

864 /wsc:DerivedKeyToken/wsc:Offset

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

872 /wsc:DerivedKeyToken/wsc:Length

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

877 /wsc:DerivedKeyToken/wsc:Label

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

881 /wsc:DerivedKeyToken/wsc:Nonce

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

886

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

- 888 • The offset is 0
- 889 • The length is 32 bytes (256 bits)

890

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

## 893 7.2 Examples

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

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

```

...
</S11:Header>
<S11:Body>
...
</S11:Body>
</S11:Envelope>

```

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

```

<wsc:DerivedKeyToken xmlns:wsc="..." xmlns:wsse="...">
  <wsse:SecurityTokenReference>
    <wsse:Reference URI="#ctx1"/>
  </wsse:SecurityTokenReference>
  <wsc:Generation>2</wsc:Generation>
</wsc:DerivedKeyToken>

```

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

```

<wsc:DerivedKeyToken xmlns:wsc="...">
  <wsc:Properties>
    <wsc:Name>.../derivedKeySource</wsc:Name>
    <wsc:Label>NewLabel</wsc:Label>
    <wsc:Nonce>FHFE...</wsc:Nonce>
  </wsc:Properties>
  <wsc:Generation>3</wsc:Generation>
</wsc:DerivedKeyToken>

```

```

<wsc:DerivedKeyToken wsu:Id="newKey" xmlns:wsc="..." xmlns:wsse="..." >
  <wsse:SecurityTokenReference>
    <wsse:Reference URI=".../derivedKeySource"/>
  </wsse:SecurityTokenReference>
  <wsc:Generation>0</wsc:Generation>
</wsc:DerivedKeyToken>

```

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

## 7.3 Implied Derived Keys

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

Consequently, the following two illustrations are functionally equivalent:

```

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

```

This is functionally equivalent to the following:

```

1006     <wsse:Security xmlns:wsc="..." xmlns:wsse="..." xmlns:xx="..."
1007     xmlns:ds="..." xmlns:wsu="...">
1008         <xx:MyToken wsu:Id="base">...</xx:MyToken>
1009         <ds:Signature>
1010             ...
1011         <ds:KeyInfo>
1012             <wsse:SecurityTokenReference wsc:Nonce="...">
1013                 <wsse:Reference URI="#base"/>
1014             </wsse:SecurityTokenReference>
1015         </ds:KeyInfo>
1016     </ds:Signature>
1017 </wsse:Security>

```

---

## 8 Associating a Security Context

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

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.

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

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

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

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

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

```
<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>
```

```
1067     ...
1068 </S11:Header>
1069 <S11:Body wsu:Id="req">
1070   <xx:Custom xmlns:xx="http://example.com/custom" xmlns:wsse="...">
1071     ...
1072     <wsse:SecurityTokenReference>
1073       <wsse:Reference URI="uuid:...UUID2..." />
1074     </wsse:SecurityTokenReference>
1075   </xx:Custom>
1076 </S11:Body>
1077 </S11:Envelope>
```

---

## 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



---

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

---

## 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.

---

## A. Sample Usages

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

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

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

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

### A.1 Anonymous SCT

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

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

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

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

A slight variation on this is as follows:

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

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

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

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

## A.2 Mutual Authentication SCT

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

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

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

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

---

## B. Token Discovery Using RST/RSTR

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

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

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

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

---

## C. Acknowledgements

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

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

Steve Anderson, OpenNetwork  
Jeff Bohren, OpenNetwork  
Toufic Boubez, Layer 7  
Marc Chanliau, Computer Associates  
Giovanni Della-Libera, Microsoft  
Brendan Dixon, Microsoft  
Praerit Garg, Microsoft  
Martin Gudgin (Editor), Microsoft  
Satoshi Hada, IBM  
Phillip Hallam-Baker, VeriSign  
Maryann Hondo, IBM  
Chris Kaler, Microsoft  
Hal Lockhart, BEA  
Robin Martherus, Oblix  
Hiroshi Maruyama, IBM  
Anthony Nadalin (Editor), IBM  
Nataraj Nagaratnam, IBM  
Andrew Nash, Reactivity  
Rob Philpott, RSA Security  
Darren Platt, Ping Identity  
Hemma Prafullchandra, VeriSign  
Maneesh Sahu, Actional  
John Shewchuk, Microsoft  
Dan Simon, Microsoft  
Davanum Srinivas, Computer Associates  
Elliot Waingold, Microsoft  
David Waite, Ping Identity  
Doug Walter, Microsoft  
Riaz Zolfonoon, RSA Security

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

Paula Austel, IBM  
Keith Ballinger, Microsoft  
John Brezak, Microsoft  
Tony Cowan, IBM  
HongMei Ge, Microsoft  
Slava Kavsan, RSA Security  
Scott Konersmann, Microsoft  
Leo Laferriere, Computer Associates  
Paul Leach, Microsoft  
Richard Levinson, Computer Associates  
John Linn, RSA Security  
Michael McIntosh, IBM  
Steve Millet, Microsoft

1262 Birgit Pfitzmann, IBM  
1263 Fumiko Satoh, IBM  
1264 Keith Stobie, Microsoft  
1265 T.R. Vishwanath, Microsoft  
1266 Richard Ward, Microsoft  
1267 Hervey Wilson, Microsoft  
1268 **TC Members during the development of this specification:**  
1269 Don Adams, Tibco Software Inc.  
1270 Jan Alexander, Microsoft Corporation  
1271 Steve Anderson, BMC Software  
1272 Donal Arundel, IONA Technologies  
1273 Howard Bae, Oracle Corporation  
1274 Abbie Barbir, Nortel Networks Limited  
1275 Charlton Barreto, Adobe Systems  
1276 Mighael Botha, Software AG, Inc.  
1277 Toufic Boubez, Layer 7 Technologies Inc.  
1278 Norman Brickman, Mitre Corporation  
1279 Melissa Brumfield, Booz Allen Hamilton  
1280 Geoff Bullen, Microsoft Corporation  
1281 Lloyd Burch, Novell  
1282 Scott Cantor, Internet2  
1283 Greg Carpenter, Microsoft Corporation  
1284 Steve Carter, Novell  
1285 Ching-Yun (C.Y.) Chao, IBM  
1286 Martin Chapman, Oracle Corporation  
1287 Kate Cherry, Lockheed Martin  
1288 Henry (Hyenvui) Chung, IBM  
1289 Luc Clement, Systinet Corp.  
1290 Paul Cotton, Microsoft Corporation  
1291 Glen Daniels, Sonic Software Corp.  
1292 Peter Davis, Neustar, Inc.  
1293 Martijn de Boer, SAP AG  
1294 Duane DeCouteau, Veterans Health Administration  
1295 Werner Dittmann, Siemens AG  
1296 Abdeslem DJAOUI, CCLRC-Rutherford Appleton Laboratory  
1297 Fred Dushin, IONA Technologies  
1298 Petr Dvorak, Systinet Corp.  
1299 Colleen Evans, Microsoft Corporation  
1300 Ruchith Fernando, WSO2  
1301 Mark Fussell, Microsoft Corporation  
1302 Vijay Gajjala, Microsoft Corporation  
1303 Marc Goodner, Microsoft Corporation  
1304 Hans Granqvist, VeriSign  
1305 Martin Gudgin, Microsoft Corporation

1306 Tony Gullotta, SOA Software Inc.  
1307 Jiandong Guo, Sun Microsystems  
1308 Phillip Hallam-Baker, VeriSign  
1309 Patrick Harding, Ping Identity Corporation  
1310 Heather Hinton, IBM  
1311 Frederick Hirsch, Nokia Corporation  
1312 Jeff Hodges, Neustar, Inc.  
1313 Will Hopkins, BEA Systems, Inc.  
1314 Alex Hristov, Otecia Incorporated  
1315 John Hughes, PA Consulting  
1316 Diane Jordan, IBM  
1317 Venugopal K, Sun Microsystems  
1318 Chris Kaler, Microsoft Corporation  
1319 Dana Kaufman, Forum Systems, Inc.  
1320 Paul Knight, Nortel Networks Limited  
1321 Ramanathan Krishnamurthy, IONA Technologies  
1322 Christopher Kurt, Microsoft Corporation  
1323 Kelvin Lawrence, IBM  
1324 Hubert Le Van Gong, Sun Microsystems  
1325 Jong Lee, BEA Systems, Inc.  
1326 Rich Levinson, Oracle Corporation  
1327 Tommy Lindberg, Dajeil Ltd.  
1328 Mark Little, JBoss Inc.  
1329 Hal Lockhart, BEA Systems, Inc.  
1330 Mike Lyons, Layer 7 Technologies Inc.  
1331 Eve Maler, Sun Microsystems  
1332 Ashok Malhotra, Oracle Corporation  
1333 Anand Mani, CrimsonLogic Pte Ltd  
1334 Jonathan Marsh, Microsoft Corporation  
1335 Robin Martherus, Oracle Corporation  
1336 Miko Matsumura, Infravio, Inc.  
1337 Gary McAfee, IBM  
1338 Michael McIntosh, IBM  
1339 John Merrells, Sxip Networks SRL  
1340 Jeff Mischkinsky, Oracle Corporation  
1341 Prateek Mishra, Oracle Corporation  
1342 Bob Morgan, Internet2  
1343 Vamsi Motukuru, Oracle Corporation  
1344 Raajmohan Na, EDS  
1345 Anthony Nadalin, IBM  
1346 Andrew Nash, Reactivity, Inc.  
1347 Eric Newcomer, IONA Technologies



1348 Duane Nickull, Adobe Systems  
1349 Toshihiro Nishimura, Fujitsu Limited  
1350 Rob Philpott, RSA Security  
1351 Denis Pilipchuk, BEA Systems, Inc.  
1352 Darren Platt, Ping Identity Corporation  
1353 Martin Raepple, SAP AG  
1354 Nick Ragouzis, Enosis Group LLC  
1355 Prakash Reddy, CA  
1356 Alain Regnier, Ricoh Company, Ltd.  
1357 Irving Reid, Hewlett-Packard  
1358 Bruce Rich, IBM  
1359 Tom Rutt, Fujitsu Limited  
1360 Maneesh Sahu, Actional Corporation  
1361 Frank Siebenlist, Argonne National Laboratory  
1362 Joe Smith, Apani Networks  
1363 Davanum Srinivas, WSO2  
1364 David Staggs, Veterans Health Administration  
1365 Yakov Sverdlov, CA  
1366 Gene Thurston, AmberPoint  
1367 Victor Valle, IBM  
1368 Asir Vedamuthu, Microsoft Corporation  
1369 Greg Whitehead, Hewlett-Packard  
1370 Ron Williams, IBM  
1371 Corinna Witt, BEA Systems, Inc.  
1372 Kyle Young, Microsoft Corporation  
1373