
WS-Trust 1.3 Interoperability Profile

Working Draft 01 07 January 2008

Specification URIs:

urn:mace:switch.ch:doc:wst:profile:interop-200801

This Version:

<http://switch.ch/grid/support/documents/wst-interop-wd01.pdf>

Previous Version:

No previous version

Latest Version:

<http://switch.ch/grid/support/documents/>

Editors:

Chad La Joie, SWITCH

Related Work:

This document relates to the [WS-Trust] security token protocol, Liberty ID-WSF SOAP Binding [WSF-SOAP], and Web Service Security [WSS] specifications.

Abstract:

This profile defines the transmission of WS-Trust messages over the Liberty SOAP binding and profiles a baseline interoperable set of options for a WS-Trust STS.

Status:

Working Draft

Table of Contents

24	1 Introduction.....	4
25	1.1 Notation.....	4
26	1.2 Normative References.....	5
27	2 Base Protocol Requirements.....	6
28	2.1 SOAP Binding and Header Blocks.....	6
29	2.1.1 SOAP Binding.....	6
30	2.1.2 <wsa:Action> Header.....	6
31	2.1.3 <sbf:Framework> Header.....	6
32	2.1.4 Optional Header Blocks.....	6
33	2.2 Operation Independent Requirements.....	6
34	2.2.1 <wst:RequestSecurityToken> Usage.....	6
35	2.2.2 <wst:RequestSecurityTokenResponse> Usage.....	7
36	2.2.3 Message Processing.....	7
37	2.3 Extension Parameters.....	7
38	3 Issuance Operation.....	8
39	3.1 Message Requirements.....	8
40	3.1.1 <wst:RequestSecurityToken> Usage.....	8
41	3.1.2 <wst:RequestSecurityTokenResponse> Usage.....	8
42	3.1.3 Extensions Parameters.....	8
43	3.1.4 Message Processing.....	8
44	3.2 Token Delegation.....	8
45	3.2.1 Delegation Extension Parameters.....	9
46	3.2.2 Message Processing.....	9
47	3.2.3 Security Considerations.....	9
48	4 Renewal Operation.....	10
49	4.1 Message Requirements.....	10
50	4.1.1 <wst:RequestSecurityToken> Usage.....	10
51	4.1.2 <wst:RequestSecurityTokenResponse> Usage.....	10
52	4.1.3 Message Processing.....	10
53	5 Cancellation Operation.....	11
54	5.1 Message Requirements.....	11
55	5.1.1 <wst:RequestSecurityToken> Usage.....	11
56	5.1.2 <wst:RequestSecurityTokenResponse> Usage.....	11
57	5.1.3 Message Processing.....	11
58	6 Validation Operation.....	12
59	6.1 Message Requirements.....	12
60	6.1.1 <wst:RequestSecurityToken> Usage.....	12
61	6.1.2 <wst:RequestSecurityTokenResponse> Usage.....	12
62	6.1.3 Message Processing.....	12
63	7 XML Signature Profile and Processing.....	13
64	7.1 Signing Formats and Algorithms.....	13
65	7.2 References.....	13
66	7.3 Canonicalization Method.....	13
67	7.4 Transforms.....	13

68	7.5 KeyInfo.....	13
69	7.6 Signature Extension Parameters.....	13
70	8 WS-Trust XML Encryption Profile and Processing.....	14
71	8.1 Canonicalization Methods.....	14
72	8.2 Referencing.....	14
73	8.3 Encryption Parameters.....	14
74	9 WS-Trust Proof of Key Possession.....	15
75	9.1 SSL/TLS Client Authentication.....	15
76	9.2 Token Signature.....	15
77	9.3 Signature Challenge.....	15
78	9.3.1 Signing Algorithms.....	15
79	10 Message Security.....	16
80	10.1 Message Integrity.....	16
81	10.2 Message Confidentiality.....	16
82	10.3 Token Integrity.....	16
83	10.4 Token Recipient Verification.....	16
84		

1 Introduction (informative)

85

86 The WS-Trust specification employs an open content model within its request and response messages.
87 This provides maximal extensibility but also means there is a theoretically infinite number of messages
88 that could be produced and yet remain compliant with the specification.

89 This profile attempts to strike a balance between the extensibility offered by the WS-Trust specification
90 and the need to scope that functionality into a manageable set. This document defines, in section 2, a set
91 of requirements common to all WS-Trust operations (issue, renew, validate, cancel). Sections 3 – 6
92 provide operation specific requirements. The remaining sections address specific security related issues.

93 This profile does not address requirements related to specific security token types. Such requirements
94 are best expressed within correlative, token-specific, profiles. This allows new tokens to be introduced
95 without editing this document and allows this document and the profile documents to be updated without
96 affecting the other documents.

1.1 Notation

97

98 This specification uses normative text to define an extension to the SAML V2.0 metadata specification.

99 The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
100 NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as
101 described in [RFC2119]:

102 ...they MUST only be used where it is actually required for interoperation or to limit
103 behavior which has potential for causing harm (e.g., limiting retransmissions)...

104 These keywords are thus capitalized when used to unambiguously specify requirements over protocol
105 and application features and behavior that affect the interoperability and security of implementations.
106 When these words are not capitalized, they are meant in their natural-language sense.

107 Listings of XML schemas appear like this.

108 Example code listings appear like this.

109
110 Conventional XML namespace prefixes are used throughout the listings in this specification to stand for
111 their respective namespaces as follows, whether or not a namespace declaration is present in the
112 example:

Prefix	XML Namespace	Comments
ds:	http://www.w3.org/2000/09/xmldsig#	This is the XML Signature namespace [XMLSig].
sb:	urn:liberty:sb:2006-08	This is the Liberty ID-WSF 2.0 SOAP Binding namespace [WSF-SOAP].
sbf:	urn:liberty:sb	This is the Liberty ID-WSF SOAP Binding namespace [WSF-SOAP].
wsa:	http://www.w3.org/2005/08/addressing	This is the Web Services Addressing namespace defined in [WSA].
wsse:	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	This is the WS-Security extension 1.0 namespace defined in [WSS].
wst:	http://docs.oasis-open.org/ws-sx/ws-trust/200512	This is the WS-Trust 1.3 namespace defined in [WS-Trust].
wsu:	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	This is the WS-Security utility namespace defined in [WSS].

Prefix	XML Namespace	Comments
xenc:	http://www.w3.org/2001/04/xmlenc#	This is the XML Encryption namespace defined in [XMLEnc].

113 This specification uses the following typographical conventions in text: <WSTrustElement>,
114 <ns:ForeignElement>, Attribute, **Datatype**, OtherKeyword.

115 1.2 Normative References

- 116 **[C14N]** J. Boyer, *Canonical XML Version 1.0*. World Web Consortium Recommendation,
117 15 March 2001. See <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- 118 **[RFC2119]** S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF
119 RFC 2119, March 1997. See <http://www.ietf.org/rfc/rfc2119.txt>
- 120 **[WSA]** D. Box, et al. *Web Services Addressing (WS-Addressing)*. World Wide Web
121 Consortium, 10 August 2004. See [http://www.w3.org/TR/2006/REC-ws-addr-](http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/)
122 [soap-20060509/](http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/)
- 123 **[WSF-SecMech]** F. Hirsch. *Liberty ID-WSF Security Mechanisms Core*. Liberty Alliance Standard.
124 See [http://www.projectliberty.org/liberty/content/download/3478/23060/file/liberty-](http://www.projectliberty.org/liberty/content/download/3478/23060/file/liberty-idwsf-security-mechanisms-core-2.0-errata-v1.0.pdf)
125 [idwsf-security-mechanisms-core-2.0-errata-v1.0.pdf](http://www.projectliberty.org/liberty/content/download/3478/23060/file/liberty-idwsf-security-mechanisms-core-2.0-errata-v1.0.pdf)
- 126 **[WSF-SOAP]** J. Hodges, et al. *Liberty ID-WSF SOAP Binding Specification*. Liberty Alliance
127 Standard. See
128 [http://www.projectliberty.org/liberty/content/download/3483/23075/file/liberty-](http://www.projectliberty.org/liberty/content/download/3483/23075/file/liberty-idwsf-soap-binding-2.0-errata-v1.0.pdf)
129 [idwsf-soap-binding-2.0-errata-v1.0.pdf](http://www.projectliberty.org/liberty/content/download/3483/23075/file/liberty-idwsf-soap-binding-2.0-errata-v1.0.pdf)
- 130 **[WSS]** A. Nadalin, et al. *Web Services Security: SOAP Message Security 1.1 (WS-*
131 *Security 2004)*. OASIS, 1 February 2004. See [http://www.oasis-](http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)
132 [open.org/committees/download.php/16790/wss-v1.1-spec-os-](http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)
133 [SOAPMessageSecurity.pdf](http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)
- 134 **[WS-Trust]** A. Nadalin, et al. *WS-Trust 1.3*. OASIS, 19 March 2007. See [http://docs.oasis-](http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf)
135 [open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf](http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf)
- 136 **[XMLEnc]** D. Eastlake, et al. *XML Encryption Syntax and Processing*. World Wide Web
137 Consortium, 10 December 2002. See [http://www.w3.org/TR/2002/REC-xmlenc-](http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/)
138 [core-20021210/](http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/)
- 139 **[XMLSig]** D. Eastlake, et al. *XML-Signature Syntax and Processing*, World Wide Web
140 Consortium, February 2002. See <http://www.w3.org/TR/xmlsig-core/>

141 **2 Base Protocol Requirements**

142 These requirements apply to all WS-Trust operation (issue, renew, validate, cancel) messages.

143 **2.1 SOAP Binding and Header Blocks**

144 **2.1.1 SOAP Binding**

145 All WS-Trust messages SHALL be transported for the Liberty ID-WSF SOAP binding and all WS-Trust
146 messages SHALL be treated as an ordinary ID-* message, as defined by the same specification. This
147 profile only employs the request-response message exchange pattern (MEP).

146 An STS implementation based on this profile MAY implement the redirection protocol defined in [WSF-
147 SOAP] section 7.

147 **2.1.2 <wsa:Action> Header**

148 This header MUST contain the value, as defined by the WS-Trust specification, that corresponds to the
149 current WS-Trust operation.

149 **2.1.3 <sbf:Framework> Header**

150 This header MUST contain the value "2.0".

151 **2.1.4 Optional Header Blocks**

152 [WSF-SOAP], section 6, defines a number of optional SOAP header blocks. An STS implementation
153 based on this profile MAY ignore these headers.

153 This profile RECOMMENDS that an implementation support, at least, <sb:EndpointUpdate>.

154 **2.2 Operation Independent Requirements**

155 All references within a message MUST be a direct, in-document, reference. Such references include
156 <wsse:SecurityTokenReference>, and <ds:Reference>.

156 Identifiers such as <wsa:MessageID>, @wsu:Id, @Context are RECOMMENDED to be
157 cryptographically random values between 128 and 160 bytes in length. An example of this are type 4
158 UUIDs.

157 Unless otherwise specified a request or response SHALL contain only one instance of a given element.

158 **2.2.1 <wst:RequestSecurityToken> Usage**

159 /wst:RequestSecurityToken/@Context

160 This attribute is RECOMMENDED. The RECOMMENDED format is a UUID URN representing a
161 generated type 4 UUID.

161 This attribute is independent of the message ID provided by the `<wsa:MessageID>` header. This
162 attributes provides a means to correlate messages, belonging to a single action that spans multiple
163 request-response pairs.

162 `/wst:RequestSecurityToken/wst:RequestType`

163 This mandatory element is used to indicate the class of function that is being requested.

164 In addition to those defined within WS-Trust this profile defines the following additional elements and
165 attributes within the `<wst:RequestSecurityToken>`.

165 `/wst:RequestSecurityToken/@wsu:Id`

166 REQUIRED if the request is signed, OPTIONAL otherwise.

167 `/wst:RequestSecurityToken/ds:Signature`

168 This element is OPTIONAL.

169 **2.2.2 `<wst:RequestSecurityTokenResponse>` Usage**

170 This profile defines the following additional elements and attributes within the
171 `<wst:RequestSecurityTokenResponse>`.

171 `/wst:RequestSecurityTokenResponse/@wsu:Id`

172 REQUIRED if the response is signed, OPTIONAL otherwise.

173 `/wst:RequestSecurityTokenResponse/ds:Signature`

174 This element is OPTIONAL.

175 **2.2.3 Message Processing**

176 The STS receiving a WS-Trust message MUST do the following:

- 177 ● Perform the process described in [WSF-SOAP] section 5.11.2.
- 178 ● Verify the freshness of the message in reference to the time-stamp carried in the
179 `<wsse:Security>` header block.
- 180 ● Verify that the request identified by `Context` has not been processed.
- 181 ● If the message is signed the signature MUST be validated.

182 **2.3 Extension Parameters**

183 The extension parameters, defined in [WS-Trust] section 9, MAY be used within a message. These
184 extensions may be required or optional and depend on contextual factors such as carried claim,
185 requested token type, etc. An STS that receives an extension parameters that is not supported within the
186 given context SHOULD return a fault.

187 **3 Issue Operation**

188 **3.1 Message Requirements**

189 **3.1.1 <wst:RequestSecurityToken> Usage**

190 `/wst:RequestSecurityToken/wst:Claims`

191 This element is REQUIRED and MUST contain exactly one security token.

192 This token is known as the presented token.

193 `/wst:RequestSecurityToken/wst:Claims/@Dialect`

194 This attribute is REQUIRED and MUST contain the URI identifying the security token in the claim.

195 `/wst:RequestSecurityToken/wst:TokenType`

196 This element is REQUIRED and MUST be populated with the profile URI of the WS-Security token
197 type being requested.

198 **3.1.2 <wst:RequestSecurityTokenResponse> Usage**

199 `/wst:RequestSecurityTokenResponse/wst:RequestedAttachedReference`

200 This element MUST be omitted.

201 `/wst:RequestSecurityTokenResponse/wst:RequestedUnattachedReference`

202 This element MUST be omitted.

203 `/wst:RequestSecurityTokenResponse/wst:Lifetime`

204 This element SHOULD be included and SHOULD provide the creation AND expiration time of the
205 token. This allows a requester to appropriately reuse the returned token.

206 **3.1.3 Extensions Parameters**

207 `/wst:RequestSecurityToken/<wst:Participants>`

208 This extension parameter may be used, by the requester, to indicate which entities should be allowed
209 to use the requested security token.

210 **3.1.4 Message Processing**

211 The STS receiving the issuance request MUST do the following processing:

- 212 ● Perform the process described in section 2.2.3
- 213 ● Verify that the `Dialect` is supported by the STS

214 3.2 Token Delegation

215 An STS MAY support the delegation of none, some, or all supported token formats. An issuance request
216 is a request for a delegate token if, and only if:

- 217 ● The presented token is a delegatable token. A token is delegatable if:
 - 218 ○ The STS supports delegation for the token type, and
 - 219 ○ The original request for the presented token contained a <wst:Delegatable> extension
220 indicating the token should be delegatable
- 221 ● The requested security token type is identical to the presented token type

222 3.2.1 Delegation Extension Parameters

223 The follow extensions parameters influence the delegation of a token.

224 `/wst:RequestSecurityToken/wst:DelegateTo`

225 This element indicates the intended recipient of the delegated token. The entity defined by this
226 extension should be considered as an additional participant in the event that the
227 <wst:Participants> extension is also used within the request.

228 `/wst:RequestSecurityToken/wst:Delegatable`

229 This element indicates that the returned security token may be delegated to another party by the
230 recipient. That is, the returned token is delegatable.

231 3.2.2 Message Processing

232 The STS receiving the token delegation issuance request MUST do the following processing:

- 233 ● Perform the processing for a normal issuance request
- 234 ● Verify that the presented token is a security token previously issued by the STS

235 3.2.3 Security Considerations

236 The mechanisms for representing a delegate token are dependent on the token type, however each
237 token type that supports delegations SHOULD:

- 238 ● Provide a clear means for a token recipient to identify that the token is a delegate token
- 239 ● Indicate the complete delegation chain of the delegate token
- 240 ● Express the intended recipient of the delegate token, as identified by the <wst:DelegateTo>
241 parameter, with the delegate token itself
- 242 ● Employ some confidentiality mechanism within the delegate token such that sensitive
243 information, meant for the recipient of the delegate, may not be viewed by the requester or any
244 intermediaries

245 **4 Renew Operation**

246 **4.1 Message Requirements**

247 **4.1.1 <wst:RequestSecurityToken> Usage**

248 /wst:RequestSecurityToken/wst:TokenType

249 This element is REQUIRED and MUST be populated with the profile URI of the WS-Security token
250 type being requested.

251 /wst:RequestSecurityToken/wst:RenewTarget

252 This element MUST contain the security token to be renewed.

253 /wst:RequestSecurityToken/wst:Renewing/@OK

254 This attribute MUST contain a value of *false* or be omitted entirely.

255 **4.1.2 <wst:RequestSecurityTokenResponse> Usage**

256 This profile does not place any additional restrictions on the elements and attributes defined for the
257 <wst:RequestSecurityTokenResponse> used during an renewal operation.

258 **4.1.3 Message Processing**

259 The STS receiving the renewal request MUST do the following processing:

- 260 ● Perform the process described in section 2.2.3
- 261 ● Verify that the requester of the renewal operation is the entity to which the token was originally
262 issued, see Section 6.3 for additional information.

263

264 **5 Cancel Operation**

265 **5.1 Message Requirements**

266 **5.1.1 <wst:RequestSecurityToken> Usage**

267 /wst:RequestSecurityToken/wst:CancelTarget

268 This element MUST contain the security token to be cancelled.

269 **5.1.2 <wst:RequestSecurityTokenResponse> Usage**

270 This profile makes no additional restrictions on the elements and attributes defined for the
271 <wst:RequestSecurityTokenResponse> used during an cancellation operation.

272 **5.1.3 Message Processing**

273 The recipient of a message MUST do the following:

- 274 ● Verify that the requester of cancellation is the entity to which the token was initially issued, see
275 Section 6.3 for additional information.

276 **6 Validate Operation**

277 **6.1 Message Requirements**

278 **6.1.1 <wst:RequestSecurityToken> Usage**

279 /wst:RequestSecurityToken/wst:ValidateTarget

280 This element MUST contain the security token to be validated.

281 **6.1.2 <wst:RequestSecurityTokenResponse> Usage**

282 This profile makes no additional restrictions on the elements and attributes defined for the
283 <wst:RequestSecurityTokenResponse> used during a validation operation.

284 **6.1.3 Message Processing**

285 This basic profile requires no additional message processing rules during a validation operation.

286 **7 XML Signature Profile and Processing**

287 This profile applies to XML digital signatures [XMLSig] used within WS-Trust messages. It does not apply
288 to digital signatures which may appear within security tokens and be defined by the token's specification.

289 **7.1 Signing Formats and Algorithms**

290 Entities MUST support the use of RSA-SHA1 in accordance with the algorithm identified by
291 <http://www.w3.org/2000/09/xmlsig#rsa-sha1>.

292 **7.2 References**

293 Signatures SHALL only employ same-document references within a `<ds:referece>`.

294 **7.3 Canonicalization Method**

295 Implementations of this profile SHOULD use Exclusive Canonicalization [C14N], with or without
296 comments (<http://www.w3.org/2001/10/xml-exc-c14n#> or
297 <http://www.w3.org/2001/10/xml-exc-c14n#WithComments>), both as the canonicalization and
298 transformation method.

299 **7.4 Transforms**

300 Signatures SHOULD NOT contain transforms other than the exclusive canonicalization or enveloped
301 signature (<http://www.w3.org/2000/09/xmlsig#enveloped-signature>) transform.

302 Signature verifiers MAY reject the signature if it employs other transforms.

303 **7.5 KeyInfo**

304 This profile does not require the use of the `<ds:KeyInfo>` within the signatures, however it is
305 RECOMMENDED in order to provide information, to the recipient, which may assist in determining the
306 correct key to use to verify the signature.

307 **7.6 Signature Extension Parameters**

308 The following extensions parameters may be used to indicate the message issuer's preferences. The
309 message issuer MAY include more than one instance of a given parameter to indicate that it supports
310 multiple mechanisms. The message receiver is then free to choose from amongst those options. If the
311 receiver can not support at least one of the mechanisms listed for a given parameter it MUST reject the
312 message.

313 `../wst:SignatureAlgorithm`

314 Indicates a signature method supported by the message issuer.

315 `../wst:CanonicalizationAlgorithm`

316 Indicates a canonicalization method supported by the message issuer.

317 8 WS-Trust XML Encryption Profile and Processing

318 This profile applies to the use of XML Encryption [XMLEnc] with WS-Trust messages. It does not apply to
319 encrypted content which may appear within security tokens and be defined by the token's specification.

320 8.1 Canonicalization Methods

321 Implementations of this profile MUST support Exclusive Canonicalization [C14N], with or without
322 comments.

323 8.2 Referencing

324 All references should be in-document, URL fragments.

325 All `<xenc:EncryptedKey>` SHOULD contain a `<xenc:ReferenceList>` identifying all components
326 signed with the key and a `<xenc:CarriedKeyName>`.

327 All `<xenc:EncryptedData>` MUST contain a `<ds:KeyInfo>` which identifies the key used to encrypt
328 the data. This may be accomplished either by the inclusion of the `<xenc:EncryptedKey>` within the
329 key info or through an in-document reference to the encrypted key by the inclusion of a
330 `<ds:RetrievalMethod>` with a Type of `http://www.w3.org/2001/04/xmlenc#EncryptedKey`.

331 8.3 Encryption Parameters

332 The following extensions parameters may be used to indicate the message issuer's preferences. The
333 message issuer MAY include more than one instance of a given parameter to indicate that it supports
334 multiple mechanisms. The message receiver is then free to choose from amongst those options. If the
335 receiver can not support at least one of the mechanisms listed for a given parameter it MUST reject the
336 message.

337 `../wst:EncryptionAlgorithm`

338 Indicates an encryption algorithm supported by the message issuer.

339 `../wst:CanonicalizationAlgorithm`

340 Indicates a canonicalization algorithm supported by the message issuer.

341 `../wst:ComputedKeyAlgorithm`

342 Indicates a computed key algorithm supported by the message issuer.

343 `../wst:KeyWrapAlgorithm`

344 Indicates a key wrap algorithm supported by the message issuer.

345 `../wst:UseKey`

346 Indicates the public or symmetric key that MUST be used to encrypt content.

347 `../wst:KeyType`

348 If `<wst:UseKey>` is used it SHOULD be accompanied by this element to indicate whether the
349 provided key is a public or symmetric key. If this parameter is not used and the message receiver is
350 unable to determine the key type, it MUST reject the message.

351 **9 WS-Trust Proof of Key Possession**

352 Some tokens may require the wielder of the token prove possession of a key associated with the token.
353 The following methods MAY be used to prove such possession.

354 **9.1 SSL/TLS Client Authentication**

355 A requester may prove possessions of a private key by employing it as the client-side token of a mutually
356 authenticated SSL/TLS connection.

357 The requester and responder MUST support this method if they support SSL/TLS based connections.

358 **9.2 Token Signature**

359 A requester may prove possession of a private key by means of a enveloped signature within the WS-
360 Trust message the covers both the request and the security token. If the message contains the token
361 directly the signature MUST have one reference indicating the message. If the message does not
362 contain the token, because it is carried within the <wsse:Security> header, the signature MUST have
363 two references, one indicating the message with its reference to the security token, and one indicating
364 the <wsse:Security> that carries the security token.

360 All requester and responders MUST support this method.

361 **9.3 Signature Challenge**

362 An alternative mechanism by which a requester may prove possession of the private key is through a
363 WS-Trust signature challenge. In this model the requester issues a request containing a claim associated
364 with a private key. The token service returns a response bearing a challenge that contains material to be
365 signed. The requester signs the materials and returning the signature to the token service. The token
366 service validates the signature and, if valid, responds to the initial request.

367 A token service SHOULD support this method.

368 **9.3.1 Signing Algorithms**

369 The algorithm used to compute the signature MUST be identified by a URI and carried in the
370 /wst:SignChallengeResponse/wst:Challenge/wst:SignatureAlgorithm attribute. Entities
371 MUST support the use of RSA with SHA1, identified by the URI
372 <http://www.w3.org/2000/09/xmldsig#rsa-sha1>, and DSA with SHA1, identified by the URI
373 <http://www.w3.org/2000/09/xmldsig#dsa-sha1>, algorithms as defined in the XML Signature
374 specification.

375 **10 Message Security**

376 **10.1 Message Integrity**

377 All security token requests and responses **MUST** employ some form of message integrity checking. Use
378 SSL/TLS **MUST** be supported and, if employed, than this requirement is met. If SSL 3.0 or TLS is not
379 used then it is **RECOMMENDED** that the requests and responses be digitally signed using XML
380 Signature. If XML Signature is used it **MUST** adhere to the profile in this specification.

381 **10.2 Message Confidentiality**

382 Confidentiality on security token requests and responses is **OPTIONAL**. Use of SSL 3.0 or TLS **MUST** be
383 supported. If SSL or TLS is not used the message, or components of the message, may be encrypted by
384 means of XML Encryption. If XML Encryption is used it **MUST** adhere to the profile in this specification.

385 A confidentiality mechanism **MUST** be used to protect any Username tokens carried within a message.

386 **10.3 Token Integrity**

387 Security tokens are requested so as to be used in subsequent messages, between a client and a service,
388 to identify the entity requesting use of the service. As such special consideration should be given to
389 providing token integrity. Such mechanisms will be token type specific.

390 **10.4 Token Recipient Verification**

391 For renewal and cancellation operations the token service is required to verify that presented security
392 token is the same token that it issued to the recipient. The mechanisms for performing this verification are
393 out of scope for this profile and may vary from implementation to implementation and token type to token
394 type. An example mechanism that may work with some token types, most notably binary token types,
395 would be for the token service to maintain a map from recipients to issued tokens. When a security token
396 is presented the token service can look up the token, by the recipient's ID and do a direct comparison of
397 the presented token with the cached token.

398 11 Examples (informative)

399 This section provides example messages meant to illustrate various operations, extensions parameters,
400 etc. These examples are non-normative and if there are mistakes/contradictions between the example
401 and this profile the normative text of the profile is authoritative.

402 Namespace declarations and base64-encoded content are not shown in the examples for the sake of
403 brevity. Specific security token content is also not shown, such examples should be found in the
404 correlative token profile.

405 11.1 Basic Issuance

406 This issuance request presents a username token and requests a X.509 certificate from which delegates
407 may be derived.

```
408 <soap:Envelope>
409   <soap:Header>
410     <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
411     trust/200512/RST/Issue</wsa:Action>
412
413     <wsa:MessageID>urn:uuid:99999999-0000-0000-0000-
414     000000000000</wsa:MessageID>
415
416     <wsa:To>http://sts.example.org</wsa:To>
417
418     <sbef:Framework version="2.0" />
419
420     <wsse:Security>
421       <wsu:Timestamp>
422         <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
423       </wsu:Timestamp>
424     </wsse:Security>
425   </soap:Header>
426
427   <soap:Body>
428     <wst:RequestSecurityToken wsu:Id="1" Context="urn:uuid:00000000-0000-
429     0000-0000-000000000000">
430
431       <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
432       trust/200512/Issue</wst:RequestType>
433
434       <wst:TokenType>
435         http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
436         profile-1.0#X509v3
437       </wst:TokenType>
438
439       <wst:Claims Dialect="http://docs.oasis-open.org/wss/2004/01/oasis-
440       200401-wss-username-token-profile-1.0">
441         <!-- Username Token -->
442       </wst:Claims>
443
444       <wst:Delegatable>true</wst:Delegatable>
445
446     </wst:RequestSecurityToken>
447   </soap:Body>
448 </soap:Envelope>
```

451 This response, to the previous request, contains the X509 security token in the SOAP header and
452 referenced from the WS-Trust response. Also, because no key was provided in the request, the token
453 service has generated a new key pair, when constructing the certificate, and has returned the private key.

```
452 <soap:Envelope>
453
454   <soap:Header>
455     <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
456     trust/200512/RSTR/IssueFinal</wsa:Action>
457
458     <wsa:MessageID>urn:uuid:99999998-0000-0000-0000-
459     000000000000</wsa:MessageID>
460
461     <wsa:RelatesTo>urn:uuid:99999999-0000-0000-0000-
462     000000000000</wsa:RelatesTo>
463
464     <sbf:Framework version="2.0" />
465
466     <wsse:Security>
467       <wsse:BinarySecurityToken wsse:Id="X509SecurityToken"
468       EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
469       wss-x509-token-profile-1.0#Base64Binary">
470         <!-- Base64-encoded X509 cert -->
471       </wsse:BinarySecurityToken>
472
473       <wsu:Timestamp>
474         <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
475       </wsu:Timestamp>
476     </wsse:Security>
477   </soap:Header>
478
479   <soap:Body>
480     <wst:RequestSecurityTokenResponse wsu:Id="2" Context="urn:uuid:00000000-
481     0000-0000-0000-000000000000">
482
483       <wst:RequestedSecurityToken>
484         <wsse:SecurityTokenReference>
485           <wsse:Reference URI="#X509SecurityToken" />
486         </wsse:SecurityTokenReference>
487       </wst:RequestedSecurityToken>
488
489       <wst:RequestedProofToken>
490         <wsse:BinarySecurityToken
491         EncodingType="http://docs.oasis-open.org/ws-sx/ws-
492         trust/200512/AsymmetricKey">
493           <!-- Base64-encoded private key -->
494         </wsse:BinarySecurityToken>
495       </wst:RequestedProofToken>
496
497     </wst:RequestSecurityTokenResponse>
498   </soap:Body>
499 </soap:Envelope>
```

495 11.2 Delegate Issuance

496 This request demonstrates a request of a delegate token derived from an X.509 certificate. However, the
497 delegated token may not, itself, be used to derive another delegate. Also, in this case, a public key is
498 provided within the request so the token service will not need to generate a new key pair.

```

497 <s12:Envelope>
498
499   <soap:Header>
500     <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
501     trust/200512/RST/Issue</wsa:Action>
502
503     <wsa:MessageID>urn:uuid:99999999-0000-0000-0000-
504     000000000000</wsa:MessageID>
505
506     <wsa:To>http://sts.example.org</wsa:To>
507
508     <wsse:Security>
509       <wsu:Timestamp>
510         <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
511       </wsu:Timestamp>
512     </wsse:Security>
513   </soap:Header>
514
515   <s12:Body>
516     <wst:RequestSecurityToken wsu:Id="1" Context="urn:uuid:00000000-0000-
517     0000-0000-000000000000">
518       <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
519       trust/200512/Issue</wst:RequestType>
520
521       <wst:TokenType>
522         http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
523         profile-1.0#X509v3
524       </wst:TokenType>
525
526       <wst:Claims Dialect="http://docs.oasis-open.org/wss/2004/01/oasis-
527       200401-wss-username-token-profile-1.0">
528         <wsse:BinarySecurityToken wsse:Id="X509SecurityToken"
529         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
530         wss-x509-token-profile-1.0#Base64Binary">
531           <!-- Base64-encoded X509 cert -->
532         </wsse:BinarySecurityToken>
533       </wst:Claims>
534
535       <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-
536       trust/200512/PublicKey</wst:KeyType>
537
538       <wst:UseKey>
539         <wsse:BinarySecurityToken wsse:Id="X509SecurityToken">
540           <!-- Base64-encoded public key -->
541         </wsse:BinarySecurityToken>
542       </wst:UseKey>
543
544       <wst:DelegateTo>urn:example.org:service:a</wst:DelegateTo>
545       <wst:Delegatable>>false</wst:Delegatable>
546
547     </wst:RequestSecurityToken>
548   </s12:Body>
549 </s12:Envelope>

```

545 The response to this request contains a proxy X.509. Because a public key was provided no private key
546 is returned with the response.

```
546 <soap:Envelope>
```

```

547
548     <soap:Header>
549         <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
550         trust/200512/RSTR/IssueFinal</wsa:Action>
551
552         <wsa:MessageID>urn:uuid:99999998-0000-0000-0000-
553         000000000000</wsa:MessageID>
554
555         <wsa:RelatesTo>urn:uuid:99999999-0000-0000-0000-
556         000000000000</wsa:RelatesTo>
557
558         <sbf:Framework version="2.0" />
559
560         <wsse:Security>
561             <wsu:Timestamp>
562                 <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
563             </wsu:Timestamp>
564         </wsse:Security>
565     </soap:Header>
566
567     <soap:Body>
568         <wst:RequestSecurityTokenResponse wsu:Id="2" Context="urn:uuid:00000000-
569         0000-0000-0000-000000000000">
570
571             <wst:RequestedSecurityToken>
572                 <wsse:BinarySecurityToken
573                     EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
574                     wss-x509-token-profile-1.0#Base64Binary">
575                     <!-- Base64-encoded public key -->
576                 </wsse:BinarySecurityToken>
577             </wst:RequestedSecurityToken>
578
579         </wst:RequestSecurityTokenResponse>
580     </soap:Body>
581 </soap:Envelope>

```

578 11.3 Renewal

579 This example demonstrates a request to renew a SAML token that requires a proof of key possession.

```

580 <soap:Envelope>
581
582     <soap:Header>
583         <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
584         trust/200512/RST/Renew</wsa:Action>
585
586         <wsa:MessageID>urn:uuid:99999999-0000-0000-0000-
587         000000000000</wsa:MessageID>
588
589         <wsa:To>http://sts.example.org</wsa:To>
590
591         <sbf:Framework version="2.0" />
592
593         <wsse:Security>
594             <wsu:Timestamp>
595                 <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
596             </wsu:Timestamp>
597         </wsse:Security>
598     </soap:Header>

```

```

597
598     <soap:Body>
599         <wst:RequestSecurityToken wsu:Id="1" Context="urn:uuid:00000000-0000-
600 0000-0000-000000000000">
601
602             <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
603 trust/200512/Renew</wst:RequestType>
604
605             <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-
606 profile-1.1#SAMLV2.0</wst:TokenType>
607
608             <wst:RenewTarget>
609                 <!-- SAML 2 Assertion -->
610             </wst:RenewTarget>
611
612         </wst:RequestSecurityToken>
613     </soap:Body>
614 </soap:Envelope>

```

613 Because the SAML token uses employs the holder-of-key confirmation method the token service requires
614 the client to prove that it posses the associated private key. The client did not sign the message with the
615 private key so the token service issues a signature challenge and employs an extension parameter to
616 indicate which signature algorithms are supported.

```

614 <soap:Envelope >
615
616     <soap:Header>
617         <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
618 trust/200512/RSTR/Renew</wsa:Action>
619
620         <wsa:MessageID>urn:uuid:99999998-0000-0000-0000-
621 000000000000</wsa:MessageID>
622
623         <wsa:RelatesTo>urn:uuid:99999999-0000-0000-0000-
624 000000000000</wsa:RelatesTo>
625
626         <sbef:Framework version="2.0" />
627
628         <wsse:Security>
629             <wsu:Timestamp>
630                 <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
631             </wsu:Timestamp>
632         </wsse:Security>
633     </soap:Header>
634
635     <soap:Body>
636         <wst:RequestSecurityTokenResponse Context="urn:uuid:00000000-0000-0000-
637 0000-000000000000">
638
639             <wst:SignChallenge>
640                 <wst:Challenge>Some content to be signed</wst:Challenge>
641             </wst:SignChallenge>
642
643             <wst:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#dsa-
644 sha1</wst:SignatureAlgorithm>
645             <wst:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#rsa-
646 sha1</wst:SignatureAlgorithm>
647
648         </wst:RequestSecurityTokenResponse>
649     </soap:Body>

```

```
644
645 </soap:Envelope>
```

646 The client responds with a signature of the supplied content.

```
647 <soap:Envelope>
648
649   <soap:Header>
650     <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
651     trust/200512/RSTR/Renew</wsa:Action>
652
653     <wsa:MessageID>urn:uuid:99999997-0000-0000-0000-
654     000000000000</wsa:MessageID>
655
656     <wsa:RelatesTo>urn:uuid:99999998-0000-0000-0000-
657     000000000000</wsa:RelatesTo>
658
659     <wsa:To>http://sts.example.org</wsa:To>
660
661     <wsse:Security>
662       <wsu:Timestamp>
663         <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
664       </wsu:Timestamp>
665     </wsse:Security>
666   </soap:Header>
667
668   <soap:Body>
669     <wst:RequestSecurityTokenResponse Context="urn:uuid:00000000-0000-0000-
670     0000-000000000000">
671       <wst:SignChallengeResponse>
672         <wst:Challenge>...Base64-encoded content
673         signature...</wst:Challenge>
674         <wst:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#rsa-
675         sha1</wst:SignatureAlgorithm>
676       </wst:SignChallengeResponse>
677     </wst:RequestSecurityTokenResponse>
678   </soap:Body>
679 </soap:Envelope>
```

679 After having verified the content signature the token service responds with an updated SAML token. Note
680 that the SAML assertion issue instant has be updated but the authentication instant has remained the
681 same. Also note that the attributes within the assertion have changed, this indicates that between the
682 time the first token was issued and the new token was issued the subject's information changed.

```
680 <soap:Envelope>
681
682   <soap:Header>
683     <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
684     trust/200512/RSTR/RenewFinal</wsa:Action>
685
686     <wsa:MessageID>urn:uuid:99999996-0000-0000-0000-
687     000000000000</wsa:MessageID>
688
689     <wsa:RelatesTo>urn:uuid:99999997-0000-0000-0000-
690     000000000000</wsa:RelatesTo>
691
692     <wsa:To>http://sts.example.org</wsa:To>
693
694     <wsse:Security>
695       <wsu:Timestamp>
696         <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
697       </wsu:Timestamp>
698     </wsse:Security>
699   </soap:Header>
700
701   <soap:Body>
702     <wst:RequestSecurityTokenResponse Context="urn:uuid:00000000-0000-0000-
703     0000-000000000000">
704       <wst:SignChallengeResponse>
705         <wst:Challenge>...Base64-encoded content
706         signature...</wst:Challenge>
707         <wst:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#rsa-
708         sha1</wst:SignatureAlgorithm>
709       </wst:SignChallengeResponse>
710     </wst:RequestSecurityTokenResponse>
711   </soap:Body>
712 </soap:Envelope>
```

```

690
691     <wsse:Security>
692         <wsu:Timestamp>
693             <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
694         </wsu:Timestamp>
695     </wsse:Security>
696 </soap:Header>
697
698     <soap:Body>
699         <wst:RequestSecurityTokenResponse wsu:Id="1" Context="urn:uuid:00000000-
700 0000-0000-0000-000000000000">
701
702             <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-
703 profile-1.1#SAMLV2.0</wst:TokenType>
704
705             <wst:RequestedSecurityToken>
706                 <!-- Renewed SAML Assertion -->
707             </wst:RequestedSecurityToken>
708
709         </wst:RequestSecurityTokenResponse>
710     </soap:Body>
711 </soap:Envelope>

```

711 11.4 Cancellation

712 This request demonstrates a cancellation operation where the client is requesting that a certificate issued
713 by the token service be cancelled or revoked. Also, the client has signed the request message in order to
714 prove possession of the certificate's private key.

```

713 <soap:Envelope>
714
715     <soap:Header>
716         <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
717 trust/200512/RST/Cancel</wsa:Action>
718
719         <wsa:MessageID>urn:uuid:99999999-0000-0000-0000-
720 000000000000</wsa:MessageID>
721
722         <wsa:To>http://sts.example.org</wsa:To>
723
724         <sbef:Framework version="2.0" />
725
726         <wsse:Security>
727             <wsu:Timestamp>
728                 <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
729             </wsu:Timestamp>
730         </wsse:Security>
731     </soap:Header>
732
733     <soap:Body>
734         <wst:RequestedSecurityToken wsu:Id="rst19383"
735 Context="urn:uuid:00000000-0000-0000-0000-000000000000">
736
737             <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
738 trust/200512/Cancel</wst:RequestType>
739
740             <wst:CancelTarget>
741                 <wsse:BinarySecurityToken wsse:Id="X509SecurityToken"

```

```

738         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
739 wss-x509-token-profile-1.0#Base64Binary">
739         <!-- Base64-encoded certificate -->
740         </wsse:BinarySecurityToken>
741         </wst:CancelTarget>
742
743         <ds:Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
744         <ds:SignedInfo>
745         <ds:CanonicalizationMethod
746 Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
746         <ds:SignatureMethod
747 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
747         <ds:Reference URI="#rst19383">
748         <ds:DigestMethod
749 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
749         <ds:DigestValue><!-- Digest Value --></ds:DigestValue>
750         <ds:Transform
751 Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
751         </ds:Reference>
752         </ds:SignedInfo>
753         <ds:SignatureValue>
754         <!-- Signature Value -->
755         </ds:SignatureValue>
756         </ds:Signature>
757
758         </wst:RequestedSecurityToken>
759     </soap:Body>
760
761 </soap:Envelope>

```

762 The token service responds that the cancellation was successful.

```

763 <soap:Envelope>
764
765     <soap:Header>
766         <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
767 trust/200512/RSTR/CancelFinal</wsa:Action>
768
769         <wsa:MessageID>urn:uuid:99999998-0000-0000-0000-
769 000000000000</wsa:MessageID>
770
771         <wsa:RelatesTo>urn:uuid:99999999-0000-0000-0000-
771 000000000000</wsa:RelatesTo>
772
773         <sbef:Framework version="2.0" />
774
775         <wsse:Security>
776         <wsu:Timestamp>
776         <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
777         </wsu:Timestamp>
778         </wsse:Security>
779     </soap:Header>
780
781     <soap:Body>
782         <wst:RequestSecurityTokenResponse wsu:Id="1" Context="urn:uuid:00000000-
783 0000-0000-0000-000000000000">
783         <wst:RequestedTokenCancelled />
784         </wst:RequestSecurityTokenResponse>
785     </soap:Body>
786
787 </soap:Envelope>

```

788 11.5 Validation

789 This example demonstrates a client requesting that a token service validate a previously issued X.509
790 certificate.

```
790 <soap:Envelope>
791
792   <soap:Header>
793     <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
794     trust/200512/RST/Validate</wsa:Action>
794
795     <wsa:MessageID>urn:uuid:99999999-0000-0000-0000-
796     000000000000</wsa:MessageID>
796
797     <wsa:To>http://sts.example.org</wsa:To>
798
799     <sbef:Framework version="2.0" />
800
801     <wsse:Security>
802       <wsu:Timestamp>
803         <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
804       </wsu:Timestamp>
805     </wsse:Security>
806   </soap:Header>
807
808   <soap:Body>
809
810     <wst:RequestSecurityToken wsu:Id="1" Context="urn:uuid:00000000-0000-
811     0000-0000-000000000000">
811
812       <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
813       trust/200512/Validate</wst:RequestType>
813
814       <wst:ValidateTarget>
815         <wsse:BinarySecurityToken wsse:Id="X509SecurityToken"
816         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
817         wss-x509-token-profile-1.0#Base64Binary">
817           <!-- Base64-encoded certificate -->
818         </wsse:BinarySecurityToken>
819       </wst:ValidateTarget>
820
821     </wst:RequestSecurityToken>
822
823   </soap:Body>
824
825 </soap:Envelope>
```

826 The token service responds that the certificate is valid.

```
827 <soap:Envelope>
828
829   <soap:Header>
830     <wsa:Action>http://docs.oasis-open.org/ws-sx/ws-
831     trust/200512/RSTR/ValidateFinal</wsa:Action>
831
832     <wsa:MessageID>urn:uuid:99999998-0000-0000-0000-
833     000000000000</wsa:MessageID>
833
834     <wsa:RelatesTo>urn:uuid:99999999-0000-0000-0000-
835     000000000000</wsa:RelatesTo>
835
```

```
836     <sbf:Framework version="2.0" />
837
838     <wsse:Security>
839         <wsu:Timestamp>
840             <wsu:Created>1970-01-01T00:00:00.000</wsu:Created>
841         </wsu:Timestamp>
842     </wsse:Security>
843 </soap:Header>
844
845     <soap:Body>
846         <wst:RequestSecurityTokenResponse wsu:Id="1" Context="urn:uuid:00000000-
847 0000-0000-0000-000000000000">
848             <wst:Status>
849                 <wst:Code>http://docs.oasis-open.org/ws-sx/ws-
849 trust/200512/status/valid</wst:Code>
849             </wst:Status>
850         </wst:RequestSecurityTokenResponse>
851     </soap:Body>
852
853 </soap:Envelope>
```