



Web Services Security: SOAP Message Security

Working Draft 10, Sunday, 23 February 2003

Document identifier:

WSS: SOAP Message Security -10

Location:

TBD

Editors:

Phillip Hallam-Baker, VeriSign
Chris Kaler, Microsoft
Ronald Monzillo, Sun
Anthony Nadalin, IBM

Contributors:

TBD – Revise this list to include WSS TC contributors

Bob Atkinson, Microsoft	John Manferdelli, Microsoft
Giovanni Della-Libera, Microsoft	Hiroshi Maruyama, IBM
Satoshi Hada, IBM	Anthony Nadalin, IBM
Phillip Hallam-Baker, VeriSign	Nataraj Nagaratnam, IBM
Maryann Hondo, IBM	Hemma Prafullchandra, VeriSign
Chris Kaler, Microsoft	John Shewchuk, Microsoft
Johannes Klein, Microsoft	Dan Simon, Microsoft
Brian LaMacchia, Microsoft	Kent Tamura, IBM
Paul Leach, Microsoft	Hervey Wilson, Microsoft

Abstract:

This specification describes enhancements to the SOAP messaging to provide quality of protection through message integrity, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

This specification also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required; it is designed to be extensible (e.g. support multiple security token formats). For example, a client might provide one format for proof of identity and provide another format for proof that they have a particular business certification.

Deleted: <sp>



Deleted:

Deleted: Core Specification

Formatted

Deleted: 09

Deleted: 26

Deleted: 1

Inserted: 1

Deleted: 8

Inserted: 8

Deleted: January

Deleted: WSS-Core

Formatted: English (U.S.)

Deleted: 09

26 Additionally, this specification describes how to encode binary security tokens, a
27 framework for XML-based tokens, and describes how to include opaque encrypted keys.
28 It also includes extensibility mechanisms that can be used to further describe the
29 characteristics of the tokens that are included with a message.

30

31 **Status:**

32 This is an interim draft. Please send comments to the editors.

33

34 Committee members should send comments on this specification to the wss@lists.oasis-
35 open.org list. Others should subscribe to and send comments to the wss-
36 comment@lists.oasis-open.org list. To subscribe, visit <http://lists.oasis->
37 [open.org/ob/adm.pl](http://lists.oasis-open.org/ob/adm.pl).

38 For information on whether any patents have been disclosed that may be essential to
39 implementing this specification, and any offers of patent licensing terms, please refer to
40 the Intellectual Property Rights section of the Security Services TC web page
41 (<http://www.oasis-open.org/who/intellectualproperty.shtml>).

42 **Table of Contents**

43	1	Introduction	5	
44	1.1	Goals and Requirements	5	
45	1.1.1	Requirements.....	5	
46	1.1.2	Non-Goals	6	Deleted: 5
47	2	Notations and Terminology	7	
48	2.1	Notational Conventions.....	7	
49	2.2	Namespaces	7	
50	2.3	Terminology.....	8	
51	3	Message Protection Mechanisms	10	
52	3.1	Message Security Model.....	10	
53	3.2	Message Protection	10	
54	3.3	Invalid or Missing Claims	11	
55	3.4	Example	11	
56	4	ID References	13	
57	4.1	Id Attribute	13	
58	4.2	Id Schema	13	
59	5	Security Header.....	15	
60	6	Security Tokens	17	
61	6.1	Attaching Security Tokens	17	
62	6.1.1	Processing Rules	17	
63	6.1.2	Subject Confirmation	17	
64	6.2	User Name Token	17	
65	6.2.1	Usernames	17	
66	6.3	Binary Security Tokens	18	
67	6.3.1	Attaching Security Tokens.....	18	
68	6.3.2	Encoding Binary Security Tokens	18	
69	6.4	XML Tokens	20	Deleted: 19
70	6.4.1	Identifying and Referencing Security Tokens	20	Deleted: 19
71	7	Token References	21	
72	7.1	SecurityTokenReference Element	21	
73	7.2	Direct References	22	
74	7.3	Key Identifiers.....	23	
75	7.4	ds:KeyInfo	24	Deleted: 23
76	7.5	Key Names	24	
77	7.6	Token Reference Lookup Processing Order	24	
78	8	Signatures	26	Deleted: 25

79	8.1 Algorithms	26	Deleted: 25
80	8.2 Signing Messages	27	Deleted: 26
81	8.3 Signature Validation	28	Deleted: 26
82	8.4 Example	29	Deleted: 27
83	9 Encryption	31	Deleted: 28
84	9.1 xenc:ReferenceList	31	Deleted: 28
85	9.2 xenc:EncryptedKey	32	Deleted: 29
86	9.3 xenc:EncryptedData	33	Deleted: 30
87	9.4 Processing Rules	34	Deleted: 30
88	9.4.1 Encryption	34	Deleted: 31
89	9.4.2 Decryption	35	Deleted: 31
90	9.5 Decryption Transformation	35	Deleted: 31
91	10 Message Timestamps	36	Deleted: 33
92	10.1 Model	36	Deleted: 33
93	10.2 Timestamp Elements	36	Deleted: 33
94	10.2.1 Creation	37	Deleted: 33
95	10.2.2 Expiration	37	Deleted: 34
96	10.3 Timestamp Header	38	Deleted: 34
97	10.4 TimestampTrace Header	39	Deleted: 36
98	11 Extended Example	41	Deleted: 38
99	12 Error Handling	44	Deleted: 41
100	13 Security Considerations	46	Deleted: 42
101	14 Privacy Considerations	48	Deleted: 44
102	15 Acknowledgements	49	Deleted: 45
103	16 References	50	Deleted: 46
104	Appendix A: Utility Elements and Attributes	52	Deleted: 48
105	A.1. Identification Attribute	52	Deleted: 48
106	A.2. Timestamp Elements	52	Deleted: 48
107	A.3. General Schema Types	53	Deleted: 49
108	Appendix B: SecurityTokenReference Model	54	Deleted: 50
109	Appendix C: Revision History	58	Deleted: 54
110	Appendix D: Notices	59	Deleted: 55
111			

112

1 Introduction

113

This specification proposes a standard set of **SOAP** extensions that can be used when building secure Web services to implement message level integrity and confidentiality. This specification refers to this set of extensions as the “Web Services Security Core Language” or “WSS-Core”.

116

This specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models including PKI, Kerberos, and SSL. Specifically, this specification provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies. The token formats and semantics for using these are defined in the associated binding documents.

121

This specification provides three main mechanisms: ability to send security token as part of a message, message integrity, and message confidentiality. These mechanisms by themselves do not provide a complete security solution for Web services. Instead, this specification is a building block that can be used in conjunction with other Web service extensions and higher-level application-specific protocols to accommodate a wide variety of security models and security technologies.

127

These mechanisms can be used independently (e.g., to pass a security token) or in a tightly coupled manner (e.g., signing and encrypting a message and providing a security token path associated with the keys used for signing and encryption).

130

1.1 Goals and Requirements

131

The goal of this specification is to enable applications to conduct secure **SOAP** message exchanges.

133

This specification is intended to provide a flexible set of mechanisms that can be used to construct a range of security protocols; in other words this specification intentionally does not describe explicit fixed security protocols.

136

As with every security protocol, significant efforts must be applied to ensure that security protocols constructed using this specification are not vulnerable to any one of a wide range of attacks.

139

The focus of this specification is to describe a single-message security language that provides for message security that may assume an established session, security context and/or policy agreement.

142

The requirements to support secure message exchange are listed below.

143

1.1.1 Requirements

144

The Web services security language must support a wide variety of security models. The following list identifies the key driving requirements for this specification:

146

- Multiple security token formats

147

- Multiple trust domains

148

- Multiple signature formats

149

- Multiple encryption technologies

- 150
- End-to-end message-level security and not just transport-level security

151 **1.1.2 Non-Goals**

152 The following topics are outside the scope of this document:

- 153
- Establishing a security context or authentication mechanisms .
- 154
- Key derivation.
- 155
- Advertisement and exchange of security policy.
- 156
- How trust is established or determined.

157

2 Notations and Terminology

158

This section specifies the notations, namespaces, and terminology used in this specification.

159

2.1 Notational Conventions

160

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 RFC 2119.

161

162

163

When describing abstract data models, this specification uses the notational convention used by the XML Infoset. Specifically, abstract property names always appear in square brackets (e.g., [some property]).

164

165

166

When describing concrete XML schemas, this specification uses the notational convention of [WSS: SOAP Message Security](#). Specifically, each member of an element's [children] or

167

168

169

170

171

172

[attributes] property is described using an XPath-like notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute wildcard (<xs:anyAttribute/>).

Deleted: WS-Security

173

174

175

176

This specification is designed to work with the general SOAPmessage structure and message processing model, and should be applicable to any version of SOAP. The current SOAP 1.2 namespace URI is used herein to provide detailed examples, but there is no intention to limit the applicability of this specification to a single version of SOAP.

177

Readers are presumed to be familiar with the terms in the [Internet Security Glossary](#).

2.2 Namespaces

178

The XML namespace URIs that MUST be used by implementations of this specification are as follows (note that elements used in this specification are from various namespaces):

179

180

181

182

```
http://schemas.xmlsoap.org/ws/2002/xx/secext
http://schemas.xmlsoap.org/ws/2002/xx/utility
```

183

The following namespaces are used in this document:

184

Prefix	Namespace
S	http://www.w3.org/2001/12/soap-envelope
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#

wsse	http://schemas.xmlsoap.org/ws/2002/xx/secext
wsu	http://schemas.xmlsoap.org/ws/2002/xx/utility

185 2.3 Terminology

186 Defined below are the basic definitions for the security terminology used in this specification.

187 **Attachment** – An *attachment* is a generic term referring to additional data that travels with a
188 SOAP message, but is not part of the SOAP Envelope.

189 **Claim** – A *claim* is a declaration made by an entity (e.g. name, identity, key, group, privilege,
190 capability, etc).

191 **Claim Confirmation** – *A claim confirmation is the process of verifying that a claim applies to*
192 *an entity*

Formatted: Label Embedded,le

Formatted: Font: Italic

193 **Confidentiality** – *Confidentiality* is the property that data is not made available to
194 unauthorized individuals, entities, or processes.

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

196 **End-To_End Message Level Security** – *End-to-end message level security* is
197 established when a message that traverses multiple applications within and between business
198 entities, e.g. companies, divisions and business units, is secure over its full route through and
199 between those business entities. This includes not only messages that are initiated within the
200 entity but also those messages that originate outside the entity, whether they are Web Services
201 or the more traditional messages.

202 **Integrity** – *Integrity* is the property that data has not been modified.

203 **Message Confidentiality** - *Message Confidentiality* is a property of the message and
204 encryption is the service or mechanism by which this property of the message is provided.

205 **Message Integrity** - *Message Integrity* is a property of the message and digital signature is
206 the service or mechanism by which this property of the message is provided.

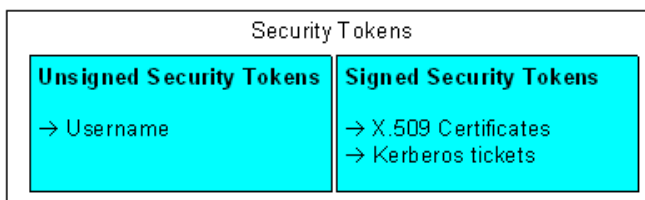
207 **Proof-of-Possession** – *Proof-of-possession* is authentication data that is provided with a
208 message to prove that the message was sent and or created by a claimed identity.

209 **Signature** - *A signature is a value computed with a cryptographic algorithm and bound*
210 *to data in such a way that intended recipients of the data can use the signature to verify that the*
211 *data has not been altered since it was signed by the signer.*

Formatted: Font: Italic

Formatted: Automatically adjust right indent when grid is defined, Space Before: 4 pt, After: 4 pt, Line spacing: single, Adjust space between Latin and Asian text, Adjust space between Asian text and numbers

212 **Security Token** – A *security token* represents a collection (one or more) of claims.



Deleted: A signature is a cryptographic binding between a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved

213

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

216 **Trust** - *Trust* is the characteristic that one entity is willing to rely upon a second entity to execute
217 a set of actions and/or to make set of assertions about a set of subjects and/or scopes.

218 **Trust Domain** – A *Trust Domain* is a security space in which the target of a request can
219 determine whether particular sets of credentials from a source satisfy the relevant security
220 policies of the target. The target may defer trust to a third party thus including the trusted third
221 party in the Trust Domain.

222

223

224

Deleted: Signature - A *signature* is a cryptographic binding between a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved. ¶

225

3 Message Protection Mechanisms

226

When securing SOAP messages, various types of threats should be considered. This includes, but is not limited to: 1) the message could be modified or read by antagonists or 2) an antagonist could send messages to a service that, while well-formed, lack appropriate security claims to warrant processing.

229

To understand these threats this specification defines a message security model.

230

3.1 Message Security Model

231

This document specifies an abstract *message security model* in terms of security tokens combined with digital signatures to protect and authenticate SOAP messages.

232

Security tokens assert claims and can be used to assert the binding between authentication secrets or keys and security identities. An authority can vouch for or endorse the claims in a security token by using its key to sign or encrypt (it is recommended to use a keyed encryption) the security token thereby enabling the authentication of the claims in the token. An X.509 certificate, claiming the binding between one's identity and public key, is an example of a signed security token endorsed by the certificate authority. In the absence of endorsement by a third party, the recipient of a security token may choose to accept the claims made in the token based on its trust of the sender of the containing message.

233

Signatures are used to verify message origin and integrity. Signatures are also used by message senders to demonstrate knowledge of the key used to confirm the claims in a security token and thus to bind their identity (and any other claims occurring in the security token) to the messages they create.

234

A signature created by a message sender to demonstrate knowledge of an authentication key is referred to as a Proof-of-Possession and may serve as a message authenticator if the signature is performed over the message.

235

236

237

238

239

It should be noted that this security model, by itself, is subject to multiple security attacks. Refer to the Security Considerations section for additional details.

240

241

Where the specification requires that the elements be "processed" this means that the element type be recognized well enough to return appropriate error if not supported.

242

243

3.2 Message Protection

244

Protecting the message content from being disclosed (confidentiality) or modified without detection (integrity) are primary security concerns. This specification provides a means to protect a message by encrypting and/or digitally signing a body, a header, an attachment, or any combination of them (or parts of them).

245

246

247

248

Message integrity is provided by leveraging XML Signature in conjunction with security tokens to ensure that messages are received without modifications. The integrity mechanisms are designed to support multiple signatures, potentially by multiple SOAP roles, and to be extensible to support additional signature formats.

249

250

251

Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of a SOAP message confidential. The encryption mechanisms are designed to support additional encryption processes and operations by multiple SOAP roles.

252

253

254

Deleted: Signatures are also used by message senders to demonstrate knowledge of the key claimed in a security token and thus to authenticate or bind their identity (and any other claims occurring in the security token) to the messages they create

264 This document defines syntax and semantics of signatures within <wsse:Security> element.
265 This document also does not specify any signature appearing outside of <wsse:Security>
266 element, if any.

267 3.3 Invalid or Missing Claims

268 The message recipient SHOULD reject a message with a signature determined to be invalid,
269 missing or unacceptable **claims** as it is an unauthorized (or malformed) message. This
270 specification provides a flexible way for the message sender to make a **claim** about the security
271 properties by associating zero or more **security tokens** with the message. An example of a
272 security **claim** is the identity of the sender; the sender can **claim** that he is Bob, known as an
273 employee of some company, and therefore he has the right to send the message.

274 3.4 Example

275 The following example illustrates the use of a username security token containing a claimed
276 security identity to establish a password derived signing key. The password is not provided in the
277 security token. The message sender combines the password with the nonce and timestamp
278 appearing in the security token to define an HMAC signing key that it then uses to sign the
279 message. The message receiver uses its knowledge of the shared secret to repeat the HMAC
280 key calculation which it uses to validate the signature and in the process confirm that the
281 message was authored by the claimed user identity. The nonce and timestamp are used in the
282 key calculation to introduce variability in the keys derived from a given password value.

```
283 (001) <?xml version="1.0" encoding="utf-8" ?>  
284 (002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
285         xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
286 (003)   <S:Header>  
287 (004)     <wsse:Security  
288             xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">  
289 (005)       <wsse:UsernameToken wsu:Id="MyID">  
290 (006)         <wsse:Username>Zoe</wsse:Username>  
291 (007)         <wsse:Nonce>FKJh...</wsse:Nonce>  
292 (008)         <wsu:Created>2001-10-13T09:00:00Z</wsu:Created>  
293 (009)       </wsse:UsernameToken>  
294 (010)       <ds:Signature>  
295 (011)         <ds:SignedInfo>  
296 (012)           <ds:CanonicalizationMethod  
297 (013)             Algorithm=  
298 (014)               "http://www.w3.org/2001/10/xml-exc-c14n#" />  
299 (015)           <ds:SignatureMethod  
300 (016)             Algorithm=  
301 (017)               "http://www.w3.org/2000/09/xmldsig#hmac-shal" />  
302 (018)           <ds:Reference URI="#MsgBody">  
303 (019)             <ds:DigestMethod  
304 (020)               Algorithm=  
305 (021)                 "http://www.w3.org/2000/09/xmldsig#sha1" />  
306 (022)             <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>  
307 (023)           </ds:Reference>  
308 (024)         </ds:SignedInfo>  
309 (025)       <ds:SignatureValue>DJBchm5gK...</ds:SignatureValue>  
310 (026)       <ds:KeyInfo>  
311 (027)         <wsse:SecurityTokenReference>
```

Formatted: English (U.S.)

Formatted: German (Germany)

Formatted: English (U.S.)

```

312      (022)          <wsse:Reference URI="#MyID" />
313      (023)          </wsse:SecurityTokenReference>
314      (024)          </ds:KeyInfo>
315      (025)          </ds:Signature>
316      (026)          </wsse:Security>
317      (027)          </S:Header>
318      (028)          <S:Body wsu:Id="MsgBody">
319      (029)          <tru:StockSymbol xmlns:tru="http://fabrikaml23.com/payloads">
320                    QQQ
321                    </tru:StockSymbol>
322      (030)          </S:Body>
323      (031)          </S:Envelope>

```

324 The first two lines start the [SOAP envelope](#). Line (003) begins the headers that are associated
325 with this [SOAP message](#).

326 Line (004) starts the [Security](#) header defined in this specification. This header contains
327 security information for an intended recipient. This element continues until line (026)

328 Lines (005) to (009) specify a [security token](#) that is associated with the message. In this case, it
329 defines *username* of the client using the [UsernameToken](#). Note that here the assumption is
330 that the service knows the password – in other words, it is a shared secret and the [Nonce](#) and
331 [Created](#) are used to generate the key

332 Lines (010) to (025) specify a digital signature. This signature ensures the [integrity](#) of the signed
333 elements. The signature uses the [XML Signature](#) specification identified by the ds namespace
334 declaration in Line (002). In this example, the signature is based on a key generated from the
335 user's password; typically stronger signing mechanisms would be used (see the [Extended](#)
336 [Example](#) later in this document).

337 Lines (011) to (018) describe what is being signed and the type of canonicalization being used.
338 Line (012) specifies how to canonicalize (normalize) the data that is being signed. Lines (014) to
339 (017) select the elements that are signed and how to digest them. Specifically, line (014)
340 indicates that the `<S:Body>` element is signed. In this example only the message body is
341 signed; typically all critical elements of the message are included in the signature (see the
342 [Extended Example](#) below).

343 Line (019) specifies the signature value of the canonicalized form of the data that is being signed
344 as defined in the [XML Signature](#) specification.

345 Lines (020) to (024) provide a *hint* as to where to find the [security token](#) associated with this
346 signature. Specifically, lines (021) to (023) indicate that the [security token](#) can be found at (pulled
347 from) the specified URL.

348 Lines (028) to (030) contain the *body* (payload) of the [SOAP](#) message.

349

350

4 ID References

351 There are many motivations for referencing other message elements such as signature
352 references or correlating signatures to security tokens. However, because arbitrary ID attributes
353 require the schemas to be available and processed, ID attributes which can be referenced in a
354 signature are restricted to the following list:

355 ID attributes from XML Signature

356 ID attributes from XML Encryption

357 wsu:Id global attribute described below

358 In addition, when signing a part of an envelope such as the body, it is RECOMMENDED that an
359 ID reference is used instead of a more general transformation, especially [XPath](#). This is to
360 simplify processing.

4.1 Id Attribute

362 There are many situations where elements within [SOAP](#) messages need to be referenced. For
363 example, when signing a SOAP message, selected elements are included in the scope of the
364 signature. [XML Schema Part 2](#) provides several built-in data types that may be used for
365 identifying and referencing elements, but their use requires that consumers of the SOAP
366 message either to have or be able to obtain the schemas where the identity or reference
367 mechanisms are defined. In some circumstances, for example, intermediaries, this can be
368 problematic and not desirable.

369 Consequently a mechanism is required for identifying and referencing elements, based on the
370 SOAP foundation, which does not rely upon complete schema knowledge of the context in which
371 an element is used. This functionality can be integrated into SOAP processors so that elements
372 can be identified and referred to without dynamic schema discovery and processing.

373 This section specifies a namespace-qualified global attribute for identifying an element which can
374 be applied to any element that either allows arbitrary attributes or specifically allows a particular
375 attribute.

4.2 Id Schema

377 To simplify the processing for intermediaries and recipients, a common attribute is defined for
378 identifying an element. This attribute utilizes the XML Schema ID type and specifies a common
379 attribute for indicating this information for elements.

380 The syntax for this attribute is as follows:

```
381 <anyElement wsu:Id="...">...</anyElement>
```

382 The following describes the attribute illustrated above:

383 `../@wsu:Id`

384 This attribute, defined as type `xsd:ID`, provides a well-known attribute for specifying the
385 local ID of an element.

386 Two `wsu:Id` attributes within an XML document MUST NOT have the same value.
387 Implementations MAY rely on XML Schema validation to provide rudimentary enforcement for

388 intra-document uniqueness. However, applications SHOULD NOT rely on schema validation
389 alone to enforce uniqueness.

390 This specification does not specify how this attribute will be used and it is expected that other
391 specifications MAY add additional semantics (or restrictions) for their usage of this attribute.

392 The following example illustrates use of this attribute to identify an element:

```
393 <x:myElement wsu:Id="ID1" xmlns:x="..."  
394 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility"/>
```

395 Conformant processors that do support XML Schema MUST treat this attribute as if it was
396 defined using a global attribute declaration.

397 Conformant processors that do not support dynamic XML Schema or DTDs discovery and
398 processing are strongly encouraged to integrate this attribute definition into their parsers. That is,
399 to treat this attribute information item as if its PSVI has a [type definition] which {target
400 namespace} is "http://www.w3.org/2001/XMLSchema" and which {name} is "Id." Doing so
401 allows the processor to inherently know *how* to process the attribute without having to locate and
402 process the associated schema. Specifically, implementations MAY support the value of the
403 `wsu:Id` as the valid identifier for use as an [XPointer](#) shorthand pointer for interoperability with
404 XML Signature references.

405

5 Security Header

406
407
408
409
410
411

The `<wsse:Security>` header block provides a mechanism for attaching security-related information targeted at a specific recipient in a form of a [SOAP role](#). This MAY be either the ultimate recipient of the message or an intermediary. Consequently, elements of this type MAY be present multiple times in a [SOAP](#) message. An intermediary on the message path MAY add one or more new sub-elements to an existing `<wsse:Security>` header block if they are targeted for its [SOAP](#) node or it MAY add one or more new headers for additional targets.

412
413
414
415
416
417
418

As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted for separate recipients. However, only one `<wsse:Security>` header block MAY omit the `S:role` attribute and no two `<wsse:Security>` header blocks MAY have the same value for `S:role`. Message security information targeted for different recipients MUST appear in different `<wsse:Security>` header blocks. The `<wsse:Security>` header block without a specified `S:role` MAY be consumed by anyone, but MUST NOT be removed prior to the final destination or endpoint.

419
420
421
422
423
424
425

As elements are added to the `<wsse:Security>` header block, they SHOULD be prepended to the existing elements. As such, the `<wsse:Security>` header block represents the signing and encryption steps the message sender took to create the message. This prepending rule ensures that the receiving application MAY process sub-elements in the order they appear in the `<wsse:Security>` header block, because there will be no forward dependency among the sub-elements. Note that this specification does not impose any specific order of processing the sub-elements. The receiving application can use whatever order is required.

426
427
428
429

When a sub-element refers to a key carried in another sub-element (for example, a signature sub-element that refers to a binary security token sub-element that contains the [X.509](#) certificate used for the signature), the key-bearing security token SHOULD be prepended to the key-using sub-element being added, so that the key material appears before the key-using sub-element.

430

The following illustrates the syntax of this header:

431
432
433
434
435
436
437
438
439
440

```
<S:Envelope>
  <S:Header>
    ...
    <wsse:Security S:role="..." S:mustUnderstand="...">
      ...
    </wsse:Security>
    ...
  </S:Header>
  ...
</S:Envelope>
```

441

The following describes the attributes and elements listed in the example above:

442

/wsse:Security

443

This is the header block for passing security-related message information to a recipient.

444

/wsse:Security/@S:role

445

This attribute allows a specific [SOAP](#) role to be identified. This attribute is optional

446

however, no two instances of the header block may omit a role or specify the same role.

447 */wsse:Security/{any}*
448 This is an extensibility mechanism to allow different (extensible) types of security
449 information, based on a schema, to be passed.
450 */wsse:Security/@{any}*
451 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
452 added to the header.
453 All compliant implementations MUST be able to process a `<wsse:Security>` element.
454 All compliant implementations MUST declare which profiles they support and MUST be able to
455 process a `<wsse:Security>` element including any sub-elements which may be defined by that
456 profile.
457 The next few sections outline elements that are expected to be used within the
458 `<wsse:Security>` header.

6 Security Tokens

459

460 This chapter specifies some different types of security tokens and how they SHALL be attached
461 to messages.

6.1 Attaching Security Tokens

462 This specification defines the `<wsse:Security>` header as a mechanism for conveying security
463 information with and about a SOAP message. This header is, by design, extensible to support
464 many types of security information.
465

466 For security tokens based on XML, the extensibility of the `<wsse:Security>` header allows for
467 these security tokens to be directly inserted into the header.

6.1.1 Processing Rules

468 This specification describes the processing rules for using and processing XML Signature and
469 XML Encryption. These rules MUST be followed when using any type of security token. Note
470 that this does NOT mean that security tokens MUST be signed or encrypted – only that if
471 signature or encryption is used in conjunction with security tokens, they MUST be used in a way
472 that conforms to the processing rules defined by this specification.
473

6.1.2 Subject Confirmation

474 ~~This specification does not dictate if and how claim confirmation must be done; however, it does
475 define how signatures may be used and associated with security tokens (by referencing the
476 security tokens from the signature) as a form of claim confirmation.~~

Deleted: This specification does not dictate if and how subject confirmation must be done, however, it does define how signatures can be used and associated with security tokens (by referencing them in the signature) as a form of Proof-of-Possession

6.2 User Name Token

6.2.1 Usernames

480 The `<wsse:UsernameToken>` element is introduced as a way of providing a username. This
481 element is optionally included in the `<wsse:Security>` header.

482 The following illustrates the syntax of this element:

```
483 <wsse:UsernameToken wsu:Id="...">  
484   <wsse:Username>...</wsse:Username>  
485 </wsse:UsernameToken>
```

Formatted: English (U.S.)

486 The following describes the attributes and elements listed in the example above:

487 */wsse:UsernameToken*

488 This element is used to represent a claimed identity.

489 */wsse:UsernameToken/@wsu:Id*

490 A string label for this security token.

491 */wsse:UsernameToken/Username*

492 This required element specifies the claimed identity.
493 `/wsse:UsernameToken/Username/@{any}`
494 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
495 the `<wsse:Username>` element.
496 `/wsse:UsernameToken/{any}`
497 This is an extensibility mechanism to allow different (extensible) types of security
498 information, based on a schema, to be passed.
499 `/wsse:UsernameToken/@{any}`
500 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
501 added to the `UsernameToken`.

502 All compliant implementations MUST be able to process a `<wsse:UsernameToken>` element.

503 The following illustrates the use of this:

```
504 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
505           xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">  
506   <S:Header>  
507     ...  
508     <wsse:Security>  
509       <wsse:UsernameToken >  
510         <wsse:Username>Zoe</wsse:Username>  
511       </wsse:UsernameToken>  
512     </wsse:Security>  
513     ...  
514   </S:Header>  
515   ...  
516 </S:Envelope>  
517
```

518 **6.3 Binary Security Tokens**

519 **6.3.1 Attaching Security Tokens**

520 For binary-formatted security tokens, this specification provides a
521 `<wsse:BinarySecurityToken>` element that can be included in the `<wsse:Security>`
522 header block.

523

524 **6.3.2 Encoding Binary Security Tokens**

525 Binary security tokens (e.g., [X.509](#) certificates and [Kerberos](#) tickets) or other non-XML formats
526 require a special encoding format for inclusion. This section describes a basic framework for
527 using binary security tokens. Subsequent specifications MUST describe the rules for creating
528 and processing specific binary security token formats.

529 The `<wsse:BinarySecurityToken>` element defines two attributes that are used to interpret
530 it. The `ValueType` attribute indicates what the security token is, for example, a [Kerberos](#) ticket.
531 The `EncodingType` tells how the security token is encoded, for example `Base64Binary`.

532 The following is an overview of the syntax:

```

533 <wsse:BinarySecurityToken wsu:Id=...
534     EncodingType=...
535     ValueType=.../>

```

536 The following describes the attributes and elements listed in the example above:

537 */wsse:BinarySecurityToken*

538 This element is used to include a binary-encoded security token.

539 */wsse:BinarySecurityToken/@wsu:Id*

540 An optional string label for this [security token](#).

541 */wsse:BinarySecurityToken/@ValueType*

542 The `ValueType` attribute is used to indicate the "value space" of the encoded binary data (e.g. an [X.509](#) certificate). The `ValueType` attribute allows a qualified name that defines the value type and space of the encoded binary data. This attribute is extensible using [XML namespaces](#). Subsequent specifications MUST define the `ValueType` value for the tokens that they define.

547 */wsse:BinarySecurityToken/@EncodingType*

548 The `EncodingType` attribute is used to indicate, using a QName, the encoding format of the binary data (e.g., `wsse:Base64Binary`). A new attribute is introduced, as there are issues with the current schema validation tools that make derivations of mixed simple and complex types difficult within [XML Schema](#). The `EncodingType` attribute is interpreted to indicate the encoding format of the element. The following encoding formats are pre-defined:

QName	Description
<code>wsse:Base64Binary</code>	XML Schema base 64 encoding

554 */wsse:BinarySecurityToken/@{any}*

555 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added.

557 All compliant implementations MUST be able to support a `<wsse:BinarySecurityToken>` element.

559 When a `<wsse:BinarySecurityToken>` is included in a signature—that is, it is referenced from a `<ds:Signature>` element—care should be taken so that the canonicalization algorithm (e.g., [Exclusive XML Canonicalization](#)) does not allow unauthorized replacement of namespace prefixes of the QNames used in the attribute or element values. In particular, it is RECOMMENDED that these namespace prefixes be declared within the `<wsse:BinarySecurityToken>` element if this token does not carry the validating key (and consequently it is not cryptographically bound to the [signature](#)). For example, if we wanted to sign the previous example, we need to include the consumed namespace definitions.

567 In the following example, a custom `ValueType` is used. Consequently, the namespace definition for this `ValueType` is included in the `<wsse:BinarySecurityToken>` element. Note that the definition of `wsse` is also included as it is used for the encoding type and the element.

```

570 <wsse:BinarySecurityToken

```

```
571     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext "
572     wsu:Id="myToken"
573     ValueType="x:MyType" xmlns:x="http://www.fabrikaml23.com/x"
574     EncodingType="wsse:Base64Binary">
575     MIEZzCCA9CgAwIBAgIQEmtJZc0...
576 </wsse:BinarySecurityToken>
```

577 **6.4 XML Tokens**

578 This section presents the basic principles and framework for using XML-based security tokens.
579 Subsequent specifications describe rules and processes for specific XML-based security token
580 formats.

581

582 **6.4.1 Identifying and Referencing Security Tokens**

583 This specification also defines multiple mechanisms for identifying and referencing security
584 tokens using the *wsu:id* attribute and the `<wsse:SecurityTokenReference>` element (as well
585 as some additional mechanisms). Please refer to the specific binding documents for the
586 appropriate reference mechanism. However, specific extensions MAY be made to the
587 `wsse:SecurityTokenReference` element.

588

589

7 Token References

590

591 This chapter discusses and defines mechanisms for referencing security tokens.

7.1 SecurityTokenReference Element

592

593 A [security token](#) conveys a set of [claims](#). Sometimes these claims reside somewhere else and
594 need to be "pulled" by the receiving application. The `<wsse:SecurityTokenReference>`
595 element provides an extensible mechanism for referencing [security tokens](#).

596 This element provides an open content model for referencing security tokens because not all
597 tokens support a common reference pattern. Similarly, some token formats have closed
598 schemas and define their own reference mechanisms. The open content model allows
599 appropriate reference mechanisms to be used when referencing corresponding token types.

600 [If a SecurityTokenReference used outside of the <Security> header block the meaning of the](#)
601 [response and/or processing rules of the resulting reference are MUST be specified by the](#)
602 [containing element and are out of scope of this specification.](#)

Deleted: The usage of

Deleted: is unspecified

Deleted: .

603 The following illustrates the syntax of this element:

```
604 <wsse:SecurityTokenReference wsu:Id="..." >  
605   ...  
606 </wsse:SecurityTokenReference>
```

607 The following describes the elements defined above:

608 */wsse:SecurityTokenReference*

609 This element provides a reference to a security token.

610 */wsse:SecurityTokenReference/@wsu:Id*

611 A string label for this [security token](#) reference. [This identifier names the reference. This](#)
612 [attribute does not indicate the ID of what is being referenced, that is done using a](#)
613 [fragment URI in a <Reference> element within the <SecurityTokenReference>](#)
614 [element.](#)

615 */wsse:SecurityTokenReference/@wsse:Usage*

616 This optional attribute is used to type the usage of the `<SecurityToken>`. Usages are
617 specified using QNames and multiple usages MAY be specified using XML list
618 semantics.

QName	Description
TBD	TBD

619

620 */wsse:SecurityTokenReference/{any}*

621 This is an extensibility mechanism to allow different (extensible) types of security
622 references, based on a schema, to be passed.

623 /wsse:SecurityTokenReference/@{any}
624 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
625 added to the header.

626 All compliant implementations MUST be able to process a
627 <wsse:SecurityTokenReference> element.

628 This element can also be used as a direct child element of <ds:KeyInfo> to indicate a hint to
629 retrieve the key information from a security token placed somewhere else. In particular, it is
630 RECOMMENDED, when using [XML Signature](#) and [XML Encryption](#), that a
631 <wsse:SecurityTokenReference> element be placed inside a <ds:KeyInfo> to reference
632 the [security token](#) used for the signature or encryption.

633 There are several challenges that implementations face when trying to interoperate. In order to
634 process the IDs and references requires the recipient to *understand* the schema. This may be an
635 expensive task and in the general case impossible as there is no way to know the "schema
636 location" for a specific namespace URI. As well, the primary goal of a reference is to uniquely
637 identify the desired token. ID references are, by definition, unique by XML. However, other
638 mechanisms such as "principal name" are not required to be unique and therefore such
639 references may be unique.

640 The following list provides a list of the specific reference mechanisms defined in [WSS: SOAP](#)
641 [Message Security](#) in preferred order (i.e., most specific to least specific):

Deleted: WS-Security

642 **Direct References** – This allows references to included tokens using URI fragments and external
643 tokens using full URIs.

644 **Key Identifiers** – This allows tokens to be referenced using an opaque value that represents the
645 token (defined by token type/profile).

646 **Key Names** – This allows tokens to be referenced using a string that matches an identity
647 assertion within the security token. This is a subset match and may result in multiple security
648 tokens that match the specified name.

649 7.2 Direct References

650 The <wsse:Reference> element provides an extensible mechanism for directly referencing
651 [security tokens](#) using URIs.

652 The following illustrates the syntax of this element:

```
653 <wsse:SecurityTokenReference wsu:Id="..." >  
654   <wsse:Reference URI="..." ValueType="..." / >  
655 </wsse:SecurityTokenReference>
```

656 The following describes the elements defined above:

657 /wsse:SecurityTokenReference/Reference

658 This element is used to identify an abstract URI location for locating a security token.

659 /wsse:SecurityTokenReference/Reference/@URI

660 This optional attribute specifies an abstract URI for where to find a security token. [If a
661 fragment is specified, then it indicates the local ID of the token being referenced.](#)

662 /wsse:SecurityTokenReference/Reference/@ValueType

663 This optional attribute specifies a QName that is used to identify the *type* of token being
664 referenced (see <wsse:BinarySecurityToken>). This specification does not define

665 any processing rules around the usage of this attribute, however, specifications for
666 individual token types MAY define specific processing rules and semantics around the
667 value of the URI and how it SHALL be interpreted. If this attribute is not present, the URI
668 SHALL be processed as a normal URI.

669 */wsse:SecurityTokenReference/Reference/{any}*

670 This is an extensibility mechanism to allow different (extensible) types of security
671 references, based on a schema, to be passed.

672 */wsse:SecurityTokenReference/Reference/@{any}*

673 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
674 added to the header.

675 The following illustrates the use of this element:

```
676 <wsse:SecurityTokenReference  
677     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">  
678   <wsse:Reference  
679     URI="http://www.fabrikam123.com/tokens/Zoe#X509token"/>  
680 </wsse:SecurityTokenReference>
```

681 7.3 Key Identifiers

682 Alternatively, if a direct reference is not used, then it is RECOMMENDED to use a key identifier to
683 specify/reference a security token instead of a ds:KeyName. The `<wsse:KeyIdentifier>`
684 element SHALL be placed in the `<wsse:SecurityTokenReference>` element to reference a
685 token using an identifier. This element SHOULD be used for all key identifiers.

686 The processing model assumes that the key identifier for a security token is constant.
687 Consequently, processing a key identifier is simply looking for a security token whose key
688 identifier matches a given specified constant.

689 The following is an overview of the syntax:

```
690 <wsse:SecurityTokenReference>  
691   <wsse:KeyIdentifier wsu:Id="..."  
692     ValueType="..."  
693     EncodingType="...">  
694     ...  
695   </wsse:KeyIdentifier>  
696 </wsse:SecurityTokenReference>
```

697 The following describes the attributes and elements listed in the example above:

698 */wsse:SecurityTokenReference/KeyIdentifier*

699 This element is used to include a binary-encoded key identifier.

700 */wsse:SecurityTokenReference/KeyIdentifier/@wsu:Id*

701 An optional string label for this identifier.

702 */wsse:SecurityTokenReference/KeyIdentifier/@ValueType*

703 The `ValueType` attribute is used to optionally indicate the type of token with the
704 specified identifier. If specified, this is a *hint* to the recipient. Any value specified for
705 binary security tokens, or any XML token element QName can be specified here. If this
706 attribute isn't specified, then the identifier applies to any type of token.

707 */wsse:SecurityTokenReference/KeyIdentifier/@EncodingType*

708 The optional `EncodingType` attribute is used to indicate, using a QName, the encoding
709 format of the binary data (e.g., `wsse:Base64Binary`). The base values defined in this
710 specification are used:

QName	Description
<code>wsse:Base64Binary</code>	XML Schema base 64 encoding (default)

711 `/wsse:SecurityTokenReference/KeyIdentifier/@{any}`

712 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
713 added.

714 7.4 ds:KeyInfo

715 The `<ds:KeyInfo>` element (from [XML Signature](#)) can be used for carrying the key information
716 and is allowed for different key types and for future extensibility. However, in this specification,
717 the use of `<wsse:BinarySecurityToken>` is the RECOMMENDED way to carry key material
718 if the key type contains binary data. Please refer to the specific binding documents for the
719 appropriate way to carry key material.

720 The following example illustrates use of this element to fetch a named key:

```
721 <ds:KeyInfo Id="..." xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
722   <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>  
723 </ds:KeyInfo>
```

724 7.5 Key Names

725 It is strongly RECOMMENDED to use key identifiers. However, if key names are used, then it is
726 strongly RECOMMENDED that `<ds:KeyName>` elements conform to the attribute names in
727 section 2.3 of RFC 2253 (this is recommended by XML Signature for `<X509SubjectName>`) for
728 interoperability.

729 Additionally, defined for e-mail addresses, SHOULD conform to RFC 822:

```
730   EmailAddress=ckaler@microsoft.com
```

731 7.6 Token Reference Lookup Processing Order

732 There are a number of mechanisms described in [XML Signature](#) and this specification
733 for referencing security tokens. To resolve possible ambiguities when more than one
734 of these reference constructs is included in a single `KeyInfo` element, the following
735 processing order SHOULD be used:

- 736 1. Resolve any `<wsse:Reference>` elements (specified within
737 `<wsse:SecurityTokenReference>`).
- 738 2. Resolve any `<wsse:KeyIdentifier>` elements (specified within
739 `<wsse:SecurityTokenReference>`).
- 740 3. Resolve any `<ds:KeyName>` elements.

- 741 4. Resolve any other <ds:KeyInfo> elements.
742 The processing stops as soon as one key has been located.

743

8 Signatures

744

Message senders may want to enable message recipients to determine whether a message was altered in transit and to verify that the claims in a particular security token apply to the sender of the message.

745

746

747

An XML signature may be used to prove that the claims in a token are confirmed by the signer.

748

Proving possession of a key associated with a token key claim supports confirming the other

749

token claims. The relying party acceptance of the claims may depend on confidence in the token

750

integrity, such as validation of an authority signature on the token. Multiple tokens may have a

751

key claim for a signature and may be referenced from the signature using a

752

SecurityTokenReference. A key claim can be an X.509 Certificate token, or a Kerberos service

753

ticket token to give two examples.

754

Because of the mutability of some SOAP headers, senders SHOULD NOT use the Enveloped

755

Signature Transform defined in XML Signature. Instead, messages SHOULD explicitly include

756

the elements to be signed. Similarly, senders SHOULD NOT use the Enveloping Signature

757

defined in XML Signature.

758

This specification allows for multiple signatures and signature formats to be attached to a

759

message, each referencing different, even overlapping, parts of the message. This is important

760

for many distributed applications where messages flow through multiple processing stages. For

761

example, a sender may submit an order that contains an orderID header. The sender signs the

762

orderID header and the body of the request (the contents of the order). When this is received by

763

the order processing sub-system, it may insert a shippingID into the header. The order sub-

764

system would then sign, at a minimum, the orderID and the shippingID, and possibly the body as

765

well. Then when this order is processed and shipped by the shipping department, a shippedInfo

766

header might be appended. The shipping department would sign, at a minimum, the shippedInfo

767

and the shippingID and possibly the body and forward the message to the billing department for

768

processing. The billing department can verify the signatures and determine a valid chain of trust

769

for the order, as well as who authorized each step in the process.

770

All compliant implementations MUST be able to support the XML Signature standard.

771

8.1 Algorithms

772

This specification builds on XML Signature and therefore has the same algorithm requirements as

773

those specified in the XML Signature specification.

774

The following table outlines additional algorithms that are strongly RECOMMENDED by this

775

specification:

Algorithm Type	Algorithm	Algorithm URI
Canonicalization	Exclusive XML Canonicalization	http://www.w3.org/2001/10/xml-exc-c14n#
Transformations	XML Decryption	http://www.w3.org/2001/04/decrypt#

Deleted: Message senders may want to enable message recipients to determine whether a message was altered in transit and to verify that a message was sent by the possessor of a particular security token.

Deleted: An XML Digital Signature can bind claims with a SOAP message body and/or headers by associating those claims with a signing key. Accepting the binding and using the claims is at the discretion of the relying party. Placing claims in one or more <SecurityTokenReference> elements that also convey the signing keys is the mechanism to create the binding of the claims. Each of these security token elements must be referenced with a <SecurityTokenReference> in the <ds:KeyInfo> element in the signature. The <SecurityTokenReference> elements can be signed, or not, depending on the relying party trust model and other requirements. ¶

	Transformation	
--	----------------	--

776 The [Exclusive XML Canonicalization](#) algorithm addresses the pitfalls of general canonicalization
777 that can occur from *leaky* namespaces with pre-existing signatures.

778 Finally, if a sender wishes to sign a message before encryption, they should use the [Decryption
779 Transformation for XML Signature](#).

780 **8.2 Signing Messages**

781 The `<wsse:Security>` header block MAY be used to carry a signature compliant with the [XML
782 Signature](#) specification within a [SOAP](#) Envelope for the purpose of signing one or more elements
783 in the [SOAP](#) Envelope. Multiple signature entries MAY be added into a single [SOAP](#) Envelope
784 within the `<wsse:Security>` header block. Senders SHOULD take care to sign all important
785 elements of the message, but care MUST be taken in creating a signing policy that will not sign
786 parts of the message that might legitimately be altered in transit.

787 [SOAP](#) applications MUST satisfy the following conditions:

788 The application MUST be capable of processing the required elements defined in the [XML
789 Signature](#) specification.

790 To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element
791 conforming to the [XML Signature](#) specification SHOULD be prepended to the existing content of
792 the `<wsse:Security>` header block. All the `<ds:Reference>` elements contained in the
793 signature SHOULD refer to a resource within the enclosing [SOAP](#) envelope, or in an attachment.

794 [XPath](#) filtering can be used to specify objects to be signed, as described in the [XML Signature](#)
795 specification. However, since the [SOAP](#) message exchange model allows intermediate
796 applications to modify the Envelope (add or delete a header block; for example), [XPath](#) filtering
797 does not always result in the same objects after message delivery. Care should be taken in using
798 [XPath](#) filtering so that there is no subsequent validation failure due to such modifications.

799 The problem of modification by intermediaries is applicable to more than just [XPath](#) processing.
800 Digital signatures, because of canonicalization and [digests](#), present particularly fragile examples
801 of such relationships. If overall message processing is to remain robust, intermediaries must
802 exercise care that their transformations do not occur within the scope of a digitally signed
803 component.

804 Due to security concerns with namespaces, this specification strongly RECOMMENDS the use of
805 the "[Exclusive XML Canonicalization](#)" algorithm or another canonicalization algorithm that
806 provides equivalent or greater protection.

807 For processing efficiency it is RECOMMENDED to have the signature added and then the
808 security token pre-pended so that a processor can read and cache the token before it is used.

Formatted: Bullets and Numbering

809 **8.3 Signing Tokens**

810 [It is often desirable to sign security tokens that are included in a message or even external to the
811 message. The XML Signature specification provides several common ways for referencing
812 information to be signed such as URIs, IDs, and XPath but some token formats may not allow
813 tokens to be referenced using URIs or IDs and XPaths may be undesirable in some situations.](#)

814 This specification allows different tokens to have their own unique reference mechanisms which
815 are specified in their profile as extensions to the <SecurityTokenReference> element. This
816 element provides a uniform referencing mechanism that is guaranteed to work with all token
817 formats. Consequently, this specification defines a new reference option for XML Signature: the
818 STR Transform.

819 This transform is specified by the URI <http://schemas.xmlsoap.org/2002/xx/STR-Transform> and
820 its contents represent a <SecurityTokenReference> element. The processing model is to
821 scan the input to the transform to locate a token(s) matching the criteria and rules defines by the
822 <SecurityTokenReference> element. Consequently, the output of the transformation is the
823 resultant sequence representing the referenced security token(s) matched

824 For references to tokens within the envelope, the URI attribute on the Reference is specified as
825 URI="". This means that the envelope is search for the relevant matching token(s). Note that a
826 more precise URI using a fragment MAY be specified. In such cases, if the reference specified in
827 the <SecurityTokenReference> element is a <Reference> element, its URI MUST be a
828 fragment. For references to tokens outside of the envelope, the URI attribute on the Reference
829 MUST be the URI of the document containing the security token. If the reference specified in the
830 <SecurityTokenReference> element is a <Reference> element, its URI MUST be either a
831 fragment or the same URI as that specified in the URI attribute of the <ds:Reference> element.

832 The following illustrates an example of this transformation which references a token contained
833 within the message envelope:

```
834 ...  
835 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">  
836   <SignedInfo>  
837     ...  
838     <Reference URI="">  
839       <Transforms>  
840         <ds:Transform  
841           Algorithm="http://schemas.xmlsoap.org/2002/xx/STR-Transform">  
842           <wsse:SecurityTokenReference>  
843             ...  
844             </wsse:SecurityTokenReference>  
845           </ds:Transform>  
846         </Transforms>  
847         <DigestMethod  
848           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />  
849         <DigestValue>...</DigestValue>  
850       </Reference>  
851     </SignedInfo>  
852     <SignatureValue></SignatureValue>  
853   </Signature>  
854 ...
```

Formatted: Bullets and Numbering

855 **8.4 Signature Validation**

856 The validation of a <ds:Signature> element inside an <wsse:Security> header block
857 SHALL fail if

- 858 • the syntax of the content of the element does not conform to this specification, or
- 859 • the validation of the signature contained in the element fails according to the core
860 validation of the XML Signature specification, or

Formatted: Bullets and Numbering

861 | • the application applying its own validation policy rejects the message for some reason
862 | (e.g., the signature is created by an untrusted key – verifying the previous two steps only
863 | performs cryptographic validation of the signature).

864 | If the validation of the signature element fails, applications MAY report the failure to the sender
865 | using the fault codes defined in Section 12 Error Handling.

Formatted: Bullets and Numbering

866 | 8.5 Example

867 | The following sample message illustrates the use of integrity and security tokens. For this
868 | example, only the message body is signed.

```
869 | <?xml version="1.0" encoding="utf-8"?>  
870 | <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
871 |           xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
872 |           xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext "  
873 |           xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">  
874 |   <S:Header>  
875 |     <wsse:Security>  
876 |       <wsse:BinarySecurityToken  
877 |         ValueType="wsse:X509v3"  
878 |         EncodingType="wsse:Base64Binary"  
879 |         wsu:Id="X509Token">  
880 |         MIIeZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...  
881 |       </wsse:BinarySecurityToken>  
882 |       <ds:Signature>  
883 |         <ds:SignedInfo>  
884 |           <ds:CanonicalizationMethod Algorithm=  
885 |             "http://www.w3.org/2001/10/xml-exc-c14n#" />  
886 |           <ds:SignatureMethod Algorithm=  
887 |             "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />  
888 |           <ds:Reference URI="#myBody">  
889 |             <ds:Transforms>  
890 |               <ds:Transform Algorithm=  
891 |                 "http://www.w3.org/2001/10/xml-exc-c14n#" />  
892 |             </ds:Transforms>  
893 |           <ds:DigestMethod Algorithm=  
894 |             "http://www.w3.org/2000/09/xmldsig#sha1" />  
895 |           <ds:DigestValue>EULddytSol...</ds:DigestValue>  
896 |         </ds:Reference>  
897 |       </ds:SignedInfo>  
898 |       <ds:SignatureValue>  
899 |       BL8jdfToEbl1/vXcMZNNjPOV...  
900 |     </ds:SignatureValue>  
901 |     <ds:KeyInfo>  
902 |       <wsse:SecurityTokenReference>  
903 |         <wsse:Reference URI="#X509Token" />  
904 |       </wsse:SecurityTokenReference>  
905 |     </ds:KeyInfo>  
906 |   </ds:Signature>  
907 | </S:Header>  
908 | <S:Body wsu:Id="myBody" >  
909 |   <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">  
910 |     QQQ  
911 |   </tru:StockSymbol>  
912 | </S:Body>
```

Formatted: German (Germany)

913
914

```
</S:Body>  
</S:Envelope>
```

9 Encryption

915

916 This specification allows encryption of any combination of body blocks, header blocks, any of
917 these sub-structures, and attachments by either a common symmetric key shared by the sender
918 and the recipient or a symmetric key carried in the message in an encrypted form.

919 In order to allow this flexibility, this specification leverages the [XML Encryption](#) standard.
920 Specifically what this specification describes is how three elements (listed below and defined in
921 [XML Encryption](#)) can be used within the `<wsse:Security>` header block. When a sender or
922 an intermediary encrypts portion(s) of a [SOAP](#) message using [XML Encryption](#) they MUST
923 prepend a sub-element to the `<wsse:Security>` header block. Furthermore, the encrypting
924 party MUST prepend the sub-element into the `<wsse:Security>` header block for the targeted
925 recipient that is expected to decrypt these encrypted portions. The combined process of
926 encrypting portion(s) of a message and adding one of these a sub-elements referring to the
927 encrypted portion(s) is called an encryption step hereafter. The sub-element should contain
928 enough information for the recipient to identify which portions of the message are to be decrypted
929 by the recipient.

930 All compliant implementations MUST be able to support the [XML Encryption](#) standard.

9.1 xenc:ReferenceList

931

932 When encrypting elements or element contents within a [SOAP](#) envelope, the
933 `<xenc:ReferenceList>` element from [XML Encryption](#) MAY be used to create a manifest of
934 encrypted portion(s), which are expressed as `<xenc:EncryptedData>` elements within the
935 envelope. An element or element content to be encrypted by this encryption step MUST be
936 replaced by a corresponding `<xenc:EncryptedData>` according to [XML Encryption](#). All the
937 `<xenc:EncryptedData>` elements created by this encryption step SHOULD be listed in
938 `<xenc:DataReference>` elements inside an `<xenc:ReferenceList>` element.

939 Although in [XML Encryption](#), `<xenc:ReferenceList>` is originally designed to be used within
940 an `<xenc:EncryptedKey>` element (which implies that all the referenced
941 `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows
942 that `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>`
943 MAY be encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>`
944 within individual `<xenc:EncryptedData>`.

945 A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the sender
946 and the recipient use a shared secret key. The following illustrates the use of this sub-element:

947

```
948 <S:Envelope  
949   xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
950   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
951   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"  
952   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">  
953   <S:Header>  
954     <wsse:Security>  
955       <xenc:ReferenceList>  
956         <xenc:DataReference URI="#bodyID"/>  
957     </xenc:ReferenceList>
```

```

957     </wsse:Security>
958 </S:Header>
959 <S:Body>
960     <xenc:EncryptedData Id="bodyID">
961         <ds:KeyInfo>
962             <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
963         </ds:KeyInfo>
964         <xenc:CipherData>
965             <xenc:CipherValue>...</xenc:CipherValue>
966         </xenc:CipherData>
967     </xenc:EncryptedData>
968 </S:Body>
969 </S:Envelope>

```

970 9.2 xenc:EncryptedKey

971 When the encryption step involves encrypting elements or element contents within a [SOAP](#)
972 envelope with a symmetric key, which is in turn to be encrypted by the recipient's key and
973 embedded in the message, <xenc:EncryptedKey> MAY be used for carrying such an
974 encrypted key. This sub-element SHOULD have a manifest, that is, an
975 <xenc:ReferenceList> element, in order for the recipient to know the portions to be
976 decrypted with this key. An element or element content to be encrypted by this encryption step
977 MUST be replaced by a corresponding <xenc:EncryptedData> according to [XML Encryption](#).
978 All the <xenc:EncryptedData> elements created by this encryption step SHOULD be listed in
979 the <xenc:ReferenceList> element inside this sub-element.

980 This construct is useful when encryption is done by a randomly generated symmetric key that is
981 in turn encrypted by the recipient's public key. The following illustrates the use of this element:

```

982 <S:Envelope
983     xmlns:S="http://www.w3.org/2001/12/soap-envelope"
984     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
985     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
986     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
987     <S:Header>
988         <wsse:Security>
989             <xenc:EncryptedKey>
990                 <xenc:EncryptionMethod Algorithm="..." />
991                 <ds:KeyInfo>
992                     <wsse:SecurityTokenReference>
993                         <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
994                             ValueType="wsse:X509v3">MIGfMa0GCSq...
995                         </wsse:KeyIdentifier>
996                     </wsse:SecurityTokenReference>
997                 </ds:KeyInfo>
998                 <xenc:CipherData>
999                     <xenc:CipherValue>...</xenc:CipherValue>
1000                 </xenc:CipherData>
1001                 <xenc:ReferenceList>
1002                     <xenc:DataReference URI="#bodyID" />
1003                 </xenc:ReferenceList>
1004             </xenc:EncryptedKey>
1005         </wsse:Security>
1006     </S:Header>
1007     <S:Body>

```



```

1008     <xenc:EncryptedData Id="bodyID">
1009         <xenc:CipherData>
1010             <xenc:CipherValue>...</xenc:CipherValue>
1011         </xenc:CipherData>
1012     </xenc:EncryptedData>
1013 </S:Body>
1014 </S:Envelope>

```

1015 While XML Encryption specifies that <xenc:EncryptedKey> elements MAY be specified in
1016 <xenc:EncryptedData> elements, this specification strongly RECOMMENDS that
1017 <xenc:EncryptedKey> elements be placed in the <wsse:Security> header.

1018 9.3 xenc:EncryptedData

1019 In some cases security-related information is provided in a purely encrypted form or non-XML
1020 attachments MAY be encrypted. The <xenc:EncryptedData> element from [XML Encryption](#)
1021 SHALL be used for these scenarios. For each part of the encrypted attachment, one encryption
1022 step is needed; that is, for each attachment to be encrypted, one <xenc:EncryptedData> sub-
1023 element MUST be added with the following rules (note that steps 2-4 applies only if MIME types
1024 are being used for attachments).

1025 The contents of the attachment MUST be replaced by the encrypted octet string.

1026 The replaced MIME part MUST have the media type application/octet-stream.

1027 The original media type of the attachment MUST be declared in the mimeType attribute of the
1028 <xenc:EncryptedData> element.

1029 The encrypted MIME part MUST be referenced by an <xenc:CipherReference> element with
1030 a URI that points to the MIME part with cid: as the scheme component of the URI.

1031 The following illustrates the use of this element to indicate an encrypted attachment:

```

1032 <S:Envelope
1033     xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1034     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1035     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
1036     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1037     <S:Header>
1038         <wsse:Security>
1039             <xenc:EncryptedData mimeType="image/png">
1040                 <ds:KeyInfo>
1041                     <wsse:SecurityTokenReference>
1042                         <xenc:EncryptionMethod Algorithm="..."/>
1043                         <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1044                             ValueType="wsse:X509v3">MIGfMa0GCSq...
1045                     </wsse:KeyIdentifier>
1046                 </wsse:SecurityTokenReference>
1047             </ds:KeyInfo>
1048             <xenc:CipherData>
1049                 <xenc:CipherReference URI="cid:image"/>
1050             </xenc:CipherData>
1051         </xenc:EncryptedData>
1052     </wsse:Security>
1053     </S:Header>
1054     <S:Body> </S:Body>
1055 </S:Envelope>

```

1056 9.4 Processing Rules

1057 Encrypted parts or attachments to the SOAP message using one of the sub-elements defined
1058 above MUST be in compliance with the XML Encryption specification. An encrypted SOAP
1059 envelope MUST still be a valid SOAP envelope. The message creator MUST NOT encrypt the
1060 <S:Envelope>, <S:Header>, or <S:Body> elements but MAY encrypt child elements of
1061 either the <S:Header> and <S:Body> elements. Multiple steps of encryption MAY be added
1062 into a single <Security> header block if they are targeted for the same recipient.

1063 When an element or element content inside a SOAP envelope (e.g. of the contents of <S:Body>)
1064 is to be encrypted, it MUST be replaced by an <xenc:EncryptedData>, according to XML
1065 Encryption and it SHOULD be referenced from the <xenc:ReferenceList> element created
1066 by this encryption step. This specification allows placing the encrypted octet stream in an
1067 attachment. For example, for an <xenc:EncryptedData> element in an <S:Body> element has
1068 <xenc:CipherReference> that refers to an attachment, then the decrypted octet stream
1069 SHALL replace the <xenc:EncryptedData>. However, if the <xenc:EncryptedData>
1070 element is located in the <Security> header block and it refers to an attachment, then the
1071 decrypted octet stream MUST replace the encrypted octet stream in the attachment.

1072 9.4.1 Encryption

1073 The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1074 this specification are listed below (note that use of <xenc:ReferenceList> is
1075 RECOMMENDED).

1076 Create a new SOAP envelope.

1077 Create a <Security> header

1078 Create an <xenc:ReferenceList> sub-element, an <xenc:EncryptedKey> sub-element, or
1079 an <xenc:EncryptedData> sub-element in the <Security> header block (note that if the
1080 SOAP "role" and "mustUnderstand" attributes are different, then a new header block may be
1081 necessary), depending on the type of encryption.

1082 Locate data items to be encrypted, i.e., XML elements, element contents within the target SOAP
1083 envelope, and attachments.

1084 Encrypt the data items as follows: For each XML element or element content within the target
1085 SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification.
1086 Each selected original element or element content MUST be removed and replaced by the
1087 resulting <xenc:EncryptedData> element. For an attachment, the contents MUST be replaced
1088 by encrypted cipher data as described in section 9.3 Signature Validation.

1089 The optional <ds:KeyInfo> element in the <xenc:EncryptedData> element MAY reference
1090 another <ds:KeyInfo> element. Note that if the encryption is based on an attached security
1091 token, then a <SecurityTokenReference> element SHOULD be added to the
1092 <ds:KeyInfo> element to facilitate locating it.

1093 Create an <xenc:DataReference> element referencing the generated
1094 <xenc:EncryptedData> elements. Add the created <xenc:DataReference> element to the
1095 <xenc:ReferenceList>.

1096 **9.4.2 Decryption**

1097 On receiving a [SOAP](#) envelope containing encryption header elements, for each encryption
1098 header element the following general steps should be processed (non-normative):

1099 Locate the `<xenc:EncryptedData>` items to be decrypted (possibly using the
1100 `<xenc:ReferenceList>`).

1101 Decrypt them as follows: For each element in the target [SOAP](#) envelope, decrypt it according to
1102 the processing rules of the [XML Encryption](#) specification and the processing rules listed above.

1103 If the decrypted data is part of an attachment and MIME types were used, then revise the MIME
1104 type of the attachment to the original MIME type (if one exists).

1105 If the decryption fails for some reason, applications MAY report the failure to the sender using the
1106 fault code defined in [Section 12 Error Handling](#).

1107 **9.5 Decryption Transformation**

1108 The ordering semantics of the `<wsse:Security>` header are sufficient to determine if
1109 signatures are over encrypted or unencrypted data. However, when a signature is included in
1110 one `<wsse:Security>` header and the encryption data is in another `<wsse:Security>`
1111 header, the proper processing order may not be apparent.

1112 If the sender wishes to sign a message that MAY subsequently be encrypted by an intermediary
1113 then the sender MAY use the Decryption Transform for XML Signature to explicitly specify the
1114 order of decryption.

1115

1116

10 Message Timestamps

1117

It is often important for the recipient to be able to determine the *freshness* of a message. In some cases, a message may be so *stale* that the recipient may decide to ignore it.

1118

1119

This specification does not provide a mechanism for synchronizing time. The assumption is either that the recipient is using a mechanism to synchronize time (e.g. NTP) or, more likely for federated applications, that they are making assessments about time based on three factors: creation time of the message, transmission checkpoints, and transmission delays and their local time.

1120

1121

1122

1123

1124

To assist a recipient in making an assessment of staleness, a requestor may wish to indicate a suggested expiration time after which the recipient should ignore the message. The specification provides XML elements by which the requestor may express the expiration time of a message, the requestor's clock time at the moment the message was created, checkpoint timestamps (when an **SOAP** role received the message) along the communication path, and the delays introduced by transmission and other factors subsequent to creation. The quality of the delays is a function of how well they reflect the actual delays (e.g., how well they reflect transmission delays).

1125

1126

1127

1128

1129

1130

1131

1132

It should be noted that this is not a protocol for making assertions or determining when, or how fast, a service produced or processed a message.

1133

1134

This specification defines and illustrates time references in terms of the *dateTimeType* defined in XML Schema. It is RECOMMENDED that all time references use this type. It is further RECOMMENDED that all references be in UTC time. If, however, other time types are used, then the *ValueType* attribute (described below) MUST be specified to indicate the data type of the time format. Requestors and receivers SHOULD NOT rely on other applications supporting time resolution finer than milliseconds. Implementations MUST NOT generate time instants that specify leap seconds.

1135

1136

1137

1138

1139

1140

1141

10.1 Model

1142

This specification provides several tools for recipients to process the expiration time presented by the requestor. The first is the **creation time**. Recipients can use this value to assess possible clock skew. However, to make some assessments, the time required to go from the requestor to the recipient may also be useful in making this assessment. Two mechanisms are provided for this. The first is that **intermediaries** may add timestamp elements indicating when they received the message. This knowledge can be useful to get a holistic view of clocks along the message path. The second is that intermediaries can specify any delays they imposed on message delivery. It should be noted that not all **delays** can be accounted for, such as wire time and parties that don't report. Recipients need to take this into account when evaluating clock skew.

1143

1144

1145

1146

1147

1148

1149

1150

1151

10.2 Timestamp Elements

1152

This specification defines the following message timestamp elements. These elements are defined for use with the `<wsu:Timestamp>` header for SOAP messages, but they can be used anywhere within the header or body that creation, expiration, and delay times are needed.

1153

1154

1155

1156 10.2.1 Creation

1157 The <wsu:Created> element specifies a creation timestamp. The exact meaning and
1158 semantics are dependent on the context in which the element is used. The syntax for this
1159 element is as follows:

```
1160 <wsu:Created ValueType="..." wsu:Id="...">...</wsu:Created>
```

1161 The following describes the attributes and elements listed in the schema above:

1162 */wsu:Created*

1163 This element's value is a creation timestamp. Its type is specified by the ValueType
1164 attribute.

1165 */wsu:Created/@ValueType*

1166 This optional attribute specifies the type of the time data. This is specified as the XML
1167 Schema type. The default value is `xsd:dateTime`.

1168 */wsu:Created/@wsu:Id*

1169 This optional attribute specifies an XML Schema ID that can be used to reference this
1170 element.

1171 10.2.2 Expiration

1172 The <wsu:Expires> element specifies the expiration time. The exact meaning and processing
1173 rules for expiration depend on the context in which the element is used. The syntax for this
1174 element is as follows:

```
1175 <wsu:Expires ValueType="..." wsu:Id="...">...</wsu:Expires>
```

1176 The following describes the attributes and elements listed in the schema above:

1177 */wsu:Expires*

1178 This element's value represents an expiration time. Its type is specified by the ValueType
1179 attribute

1180 */wsu:Expires/@ValueType*

1181 This optional attribute specifies the type of the time data. This is specified as the XML
1182 Schema type. The default value is `xsd:dateTime`.

1183 */wsu:Expires/@wsu:Id*

1184 This optional attribute specifies an XML Schema ID that can be used to reference this
1185 element.

1186 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
1187 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
1188 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
1189 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
1190 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
1191 judgment of the requestor's likely current clock time by means not described in this specification,
1192 for example an out-of-band clock synchronization protocol. The recipient may also use the
1193 creation time and the delays introduced by intermediate SOAP roles to estimate the degree of
1194 clock skew .

1195 One suggested formula for estimating clock skew is

1196 `skew = recipient's arrival time - creation time - transmission time`

1197 Transmission time may be estimated by summing the values of delay elements, if present. It
1198 should be noted that wire-time is only part of this if delays include it in estimates. Otherwise the
1199 transmission time will not reflect the on-wire time. If no delays are present, there are no special
1200 assumptions that need to be made about processing time

1201 10.3 Timestamp Header

1202 A `<wsu:Timestamp>` header provides a mechanism for expressing the creation and expiration
1203 times of a message introduced throughout the message path. Specifically, it uses the previously
1204 defined elements in the context of message creation, receipt, and processing.

1205 All times SHOULD be in UTC format as specified by the [XML Schema](#) type (`dateTime`). It should
1206 be noted that times support time precision as defined in the [XML Schema](#) specification.

1207 Multiple `<wsu:Timestamp>` headers can be specified if they are targeted at different [SOAP](#)
1208 roles. The ordering within the header is as illustrated below.

1209 The ordering of elements in this header is fixed and MUST be preserved by intermediaries.

1210 To preserve overall integrity of each `<wsu:Timestamp>` header, it is strongly RECOMMENDED
1211 that each [SOAP](#) role create or update the appropriate `<wsu:Timestamp>` header destined to
1212 itself.

1213 The schema outline for the `<wsu:Timestamp>` header is as follows:

```
1214 <wsu:Timestamp wsu:Id="...">  
1215   <wsu:Created>...</wsu:Created>  
1216   <wsu:Expires>...</wsu:Expires>  
1217   ...  
1218 </wsu:Timestamp>
```

1219 The following describes the attributes and elements listed in the schema above:

1220 */wsu:Timestamp*

1221 This is the header for indicating message timestamps.

1222 */wsu:Timestamp/Created*

1223 This represents the [creation time](#) of the message. This element is optional, but can only
1224 be specified once in a `Timestamp` header. Within the SOAP processing model, creation
1225 is the instant that the infonet is serialized for transmission. The creation time of the
1226 message SHOULD NOT differ substantially from its transmission time. The difference in
1227 time should be minimized.

1228 */wsu:Timestamp/Expires*

1229 This represents the [expiration](#) of the message. This is optional, but can appear at most
1230 once in a `Timestamp` header. Upon expiration, the requestor asserts that the message
1231 is no longer valid. It is strongly RECOMMENDED that recipients (anyone who processes
1232 this message) discard (ignore) any message that has passed its expiration. A Fault code
1233 (`wsu:MessageExpired`) is provided if the recipient wants to inform the requestor that its
1234 message was expired. A service MAY issue a Fault indicating the message has expired.

1235 */wsu:Timestamp/{any}*

1236 This is an extensibility mechanism to allow additional elements to be added to the
1237 header.

1238 */wsu:Timestamp/@wsu:Id*
1239 This optional attribute specifies an XML Schema ID that can be used to reference this
1240 element.

1241 */wsu:Timestamp/@{any}*
1242 This is an extensibility mechanism to allow additional attributes to be added to the
1243 header.

1244 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```
1245 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
1246           xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">  
1247   <S:Header>  
1248     <wsu:Timestamp>  
1249       <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>  
1250       <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>  
1251     </wsu:Timestamp>  
1252     ...  
1253   </S:Header>  
1254   <S:Body>  
1255     ...  
1256   </S:Body>  
1257 </S:Envelope>
```

1258 10.4 TimestampTrace Header

1259 A `<wsu:TimestampTrace>` header provides a mechanism for expressing the delays introduced
1260 throughout the message path. Specifically, it uses the previously defined elements in the context
1261 of message creation, receipt, and processing.

1262 All times SHOULD be in UTC format as specified by the [XML Schema](#) type (`dateTime`). It should
1263 be noted that times support time precision as defined in the [XML Schema](#) specification.

1264 Multiple `<wsu:TimestampTrace>` headers can be specified if they reference a different [SOAP](#)
1265 role.

1266 The `<wsu:Received>` element specifies a receipt timestamp with an optional processing delay.
1267 The exact meaning and semantics are dependent on the context in which the element is used.

1268 It is also strongly RECOMMENDED that each [SOAP](#) role sign its elements by referencing their
1269 ID, NOT by signing the `TimestampTrace` header as the header is mutable.

1270 The syntax for this element is as follows:

```
1271 <wsu:TimestampTrace>  
1272   <wsu:Received Role="..." Delay="..." ValueType="..."  
1273     wsu:Id="..." >...</wsu:Received>  
1274 </wsu:TimestampTrace>
```

1275 The following describes the attributes and elements listed in the schema above:

1276 */wsu:Received*

1277 This element's value is a receipt timestamp. The time specified SHOULD be a UTC
1278 format as specified by the `ValueType` attribute (default is [XML Schema](#) type `dateTime`).

1279 */wsu:Received/@Role*

1280 A required attribute, `Role`, indicates which **SOAP** role is indicating receipt. Roles MUST
 1281 include this attribute, with a value matching the role value as specified as a SOAP
 1282 intermediary.

1283 `/wsu:Received/@Delay`

1284 The value of this optional attribute is the delay associated with the **SOAP** role expressed
 1285 in milliseconds. The delay represents processing time by the Role after it received the
 1286 message, but before it forwarded to the next recipient.

1287 `/wsu:Received/@ValueType`

1288 This optional attribute specifies the type of the time data (the element value). This is
 1289 specified as the XML Schema type. If this attribute isn't specified, the default value is
 1290 `xsd:dateTime`.

1291 `/wsu:Received/@wsu:Id`

1292 This optional attribute specifies an XML Schema ID that can be used to reference this
 1293 element.

1294 The delay attribute indicates the time delay attributable to an **SOAP** role (intermediate
 1295 processor). In some cases this isn't known; for others it can be computed as *role's send time –*
 1296 *role's receipt time*.

1297 Each delay amount is indicated in units of milliseconds, without fractions. If a delay amount
 1298 would exceed the maximum value expressible in the datatype, the value should be set to the
 1299 maximum value of the datatype.

1300 The following example illustrates the use of the `<wsu:Timestamp>` header and a
 1301 `<wsu:TimestampTrace>` header indicating a processing delay of one minute subsequent to the
 1302 receipt which was two minutes after creation.

```

1303 <S:Envelope xmlns:S=" http://www.w3.org/2001/12/soap-envelope"
1304           xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
1305   <S:Header>
1306     <wsu:Timestamp>
1307       <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
1308       <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
1309     </wsu:Timestamp>
1310     <wsu:TimestampTrace>
1311       <wsu:Received Role="http://x.com/" Delay="60000">
1312         2001-09-13T08:44:00Z</wsu:Received>
1313     </wsu:TimestampTrace>
1314     ...
1315   </S:Header>
1316   <S:Body>
1317     ...
1318   </S:Body>
1319 </S:Envelope>
1320

```


1321

11 Extended Example

1322

The following sample message illustrates the use of security tokens, signatures, and encryption.

1323

For this example, the timestamp and the message body are signed prior to encryption. The

1324

decryption transformation is not needed as the signing/encryption order is specified within the

1325

<wsse:Security> header.

1326

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
(003)   <S:Header>
(004)     <wsu:Timestamp>
(005)       <wsu:Created wsu:Id="T0">
(006)         2001-09-13T08:42:00Z
(007)       </wsu:Created>
(008)     </wsu:Timestamp>
(009)     <wsse:Security>
(010)       <wsse:BinarySecurityToken
            ValueType="wsse:X509v3"
            wsu:Id="X509Token"
            EncodingType="wsse:Base64Binary">
(011)         MIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
(012)       </wsse:BinarySecurityToken>
(013)       <xenc:EncryptedKey>
(014)         <xenc:EncryptionMethod Algorithm=
            "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
(015)         <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
            ValueType="wsse:X509v3">MIGfMa0GCSq...
(016)         </wsse:KeyIdentifier>
(017)         <xenc:CipherData>
(018)           <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(019)           </xenc:CipherValue>
(020)         </xenc:CipherData>
(021)         <xenc:ReferenceList>
(022)           <xenc:DataReference URI="#enc1"/>
(023)         </xenc:ReferenceList>
(024)       </xenc:EncryptedKey>
(025)       <ds:Signature>
(026)         <ds:SignedInfo>
(027)           <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
(028)           <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
(029)           <ds:Reference URI="#T0">
(030)             <ds:Transforms>
(031)               <ds:Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
(032)             </ds:Transforms>
(033)           </ds:Reference>
(033)         </ds:SignedInfo>
(033)       </ds:Signature>
(033)     </wsse:Security>
(033)   </S:Header>
(033) </S:Envelope>
```

```

1370 (034)         <ds:DigestMethod
1371             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
1372 (035)         <ds:DigestValue>LyLsF094hPi4wPU...
1373 (036)         </ds:DigestValue>
1374 (037)         </ds:Reference>
1375 (038)         <ds:Reference URI="#body">
1376 (039)         <ds:Transforms>
1377 (040)             <ds:Transform
1378                 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1379 (041)             </ds:Transforms>
1380 (042)         <ds:DigestMethod
1381             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
1382 (043)         <ds:DigestValue>LyLsF094hPi4wPU...
1383 (044)         </ds:DigestValue>
1384 (045)         </ds:Reference>
1385 (046)         </ds:SignedInfo>
1386 (047)         <ds:SignatureValue>
1387 (048)             HplZkmFZ/2kQLXDjBchm5gK...
1388 (049)         </ds:SignatureValue>
1389 (050)         <ds:KeyInfo>
1390 (051)             <wsse:SecurityTokenReference>
1391 (052)                 <wsse:Reference URI="#X509Token"/>
1392 (053)             </wsse:SecurityTokenReference>
1393 (054)             </ds:KeyInfo>
1394 (055)         </ds:Signature>
1395 (056)     </wsse:Security>
1396 (057) </S:Header>
1397 (058) <S:Body wsu:Id="body">
1398 (059)     <xenc:EncryptedData
1399             Type="http://www.w3.org/2001/04/xmlenc#Element"
1400             wsu:Id="enc1">
1401 (060)         <xenc:EncryptionMethod
1402             Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
1403 (061)         <xenc:CipherData>
1404 (062)             <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1405 (063)             </xenc:CipherValue>
1406 (064)         </xenc:CipherData>
1407 (065)     </xenc:EncryptedData>
1408 (066) </S:Body>
1409 (067) </S:Envelope>

```

1410 Let's review some of the key sections of this example:

1411 Lines (003)-(057) contain the SOAP message headers.

1412 Lines (004)-(008) specify the timestamp information. In this case it indicates the creation time of
1413 the message.

1414 Lines (009)-(056) represent the `<wsse:Security>` header block. This contains the security-
1415 related information for the message.

1416 Lines (010)-(012) specify a [security token](#) that is associated with the message. In this case, it
1417 specifies an [X.509](#) certificate that is encoded as Base64. Line (011) specifies the actual Base64
1418 encoding of the certificate.

1419 Lines (013)-(025) specify the key that is used to encrypt the body of the message. Since this is a
1420 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
1421 encrypt the key. Lines (015)-(017) specify the name of the key that was used to encrypt the

1422 symmetric key. Lines (018)-(021) specify the actual encrypted form of the symmetric key. Lines
1423 (022)-(024) identify the encryption block in the message that uses this symmetric key. In this
1424 case it is only used to encrypt the body (Id="enc1").

1425 Lines (026)-(055) specify the digital signature. In this example, the signature is based on the
1426 [X.509](#) certificate. Lines (027)-(046) indicate what is being signed. Specifically, Line (039)
1427 references the creation timestamp and line (038) references the message body.

1428 Lines (047)-(049) indicate the actual signature value – specified in Line (042).

1429 Lines (051)-(053) indicate the key that was used for the signature. In this case, it is the [X.509](#)
1430 certificate included in the message. Line (052) provides a URI link to the Lines (010)-(012).

1431 The body of the message is represented by Lines (056)-(066).

1432 Lines (059)-(065) represent the encrypted metadata and form of the body using [XML Encryption](#).
1433 Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line
1434 (060) specifies the encryption algorithm – Triple-DES in this case. Lines (062)-(063) contain the
1435 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
1436 key as the key references this encryption – Line (023).

1437

12 Error Handling

1438 There are many circumstances where an *error* can occur while processing security information.
1439 For example:

1440 Invalid or unsupported type of security token, signing, or encryption

1441 Invalid or unauthenticated or unauthenticatable security token

1442 Invalid signature

1443 Decryption failure

1444 Referenced security token is unavailable

1445 Unsupported namespace

1446 These can be grouped into two *classes* of errors: unsupported and failure. For the case of
1447 unsupported errors, the recipient *MAY* provide a response that informs the sender of supported
1448 formats, etc. For failure errors, the recipient *MAY* choose not to respond, as this may be a form
1449 of Denial of Service (DOS) or cryptographic attack. We combine signature and encryption
1450 failures to mitigate certain types of attacks.

1451 If a failure is returned to a sender then the failure *MUST* be reported using [SOAP's Fault](#)
1452 mechanism. The following tables outline the predefined security fault codes. The "unsupported"
1453 class of errors are:

Error that occurred	faultcode
An unsupported token was provided	wsse:UnsupportedSecurityToken
An unsupported signature or encryption algorithm was used	wsse:UnsupportedAlgorithm

1454 The "failure" class of errors are:

Error that occurred	faultcode
An error was discovered processing the <wsse:Security> header.	wsse:InvalidSecurity
An invalid security token was provided	wsse:InvalidSecurityToken
The security token could not be authenticated or authorized	wsse:FailedAuthentication
The signature or decryption was invalid	wsse:FailedCheck

Referenced security token could not be retrieved

wsse:SecurityTokenUnavailable

13 Security Considerations

1455

1456 It is strongly RECOMMENDED that messages include digitally signed elements to allow message
1457 recipients to detect replays of the message when the messages are exchanged via an open
1458 network. These can be part of the message or of the headers defined from other SOAP
1459 extensions. Four typical approaches are:

1460 Timestamp

1461 Sequence Number

1462 Expirations

1463 Message Correlation

1464 This specification defines the use of XML Signature and XML Encryption in SOAP headers. As
1465 one of the building blocks for securing SOAP messages, it is intended to be used in conjunction
1466 with other security techniques. Digital signatures need to be understood in the context of other
1467 security mechanisms and possible threats to an entity.

1468 Digital signatures alone do not provide message authentication. One can record a signed
1469 message and resend it (a replay attack). To prevent this type of attack, digital signatures must be
1470 combined with an appropriate means to ensure the uniqueness of the message, such as
1471 timestamps or sequence numbers (see earlier section for additional details). The proper usage of
1472 nonce guards against replay attacks.

1473 ~~When digital signatures are used for verifying the claims pertaining to the sending entity, the~~
1474 ~~sender must demonstrate knowledge of the confirmation key. One way to achieve this is to use a~~
1475 ~~challenge-response type of protocol. Such a protocol is outside the scope of this document.~~

1476 To this end, the developers can attach timestamps, expirations, and sequences to messages.

1477 Implementers should also be aware of all the security implications resulting from the use of digital
1478 signatures in general and XML Signature in particular. When building trust into an application
1479 based on a digital signature there are other technologies, such as certificate evaluation, that must
1480 be incorporated, but these are outside the scope of this document.

1481 Requestors should use digital signatures to sign security tokens that do not include signatures (or
1482 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
1483 RECOMMENDED that all relevant and immutable message content be signed by the sender.
1484 Receivers SHOULD only consider those portions of the document that are covered by the
1485 sender's signature as being subject to the security tokens in the message. Security tokens
1486 appearing in <wssc:Security> header elements SHOULD be signed by their issuing authority
1487 so that message receivers can have confidence that the security tokens have not been forged or
1488 altered since their issuance. It is strongly RECOMMENDED that a message sender sign any
1489 <SecurityToken> elements that it is confirming and that are not signed by their issuing
1490 authority.

1491 Also, as described in XML Encryption, we note that the combination of signing and encryption
1492 over a common data item may introduce some cryptographic vulnerability. For example,
1493 encrypting digitally signed data, while leaving the digital signature in the clear, may allow plain
1494 text guessing attacks. The proper usage of nonce guards against replay attacks.

Deleted: When digital signatures are used for verifying the identity of the sending party, the sender must prove the possession of the private key. One way to achieve this is to use a challenge-response type of protocol. Such a protocol is outside the scope of this document

1495 In order to *trust* IDs and timestamps, they SHOULD be signed using the mechanisms outlined in
1496 this specification. This allows readers of the IDs and timestamps information to be certain that
1497 the IDs and timestamps haven't been forged or altered in any way. It is strongly
1498 RECOMMENDED that IDs and timestamp elements be signed.

1499 Timestamps can also be used to mitigate replay attacks. Signed timestamps MAY be used to
1500 keep track of messages (possibly by caching the most recent timestamp from a specific service)
1501 and detect replays of previous messages. It is RECOMMENDED that timestamps and nonces be
1502 cached for a given period of time, as a guideline a value of five minutes can be used as a
1503 minimum to detect replays, and that timestamps older than that given period of time set be
1504 rejected. in interactive scenarios.

1505 | When a password ([or password equivalent](#)) in a <UsernameToken> is used for authentication,
1506 the password needs to be properly protected. If the underlying transport does not provide enough
1507 protection against eavesdropping, the password SHOULD be digested as described in Section
1508 6.1.1. Even so, the password must be strong enough so that simple password guessing attacks
1509 will not reveal the secret from a captured message.

1510 In one-way message authentication, it is RECOMMENDED that the sender and the recipient re-
1511 use the elements and structure defined in this specification for proving and validating freshness of
1512 a message. It is RECOMMEND that the nonce value be unique per message (never been used
1513 as a nonce before by the sender and recipient) and use the <wsse:Nonce> element within the
1514 <wsse:Security> header. Further, the <wsu:Timestamp> header SHOULD be used with a
1515 <wsu:Created> element. It is strongly RECOMMENDED that the <wsu:Created> ,
1516 <wsse:Nonce> elements be included in the signature.

1517

14 Privacy Considerations

1518 TBD

1519

15 Acknowledgements

1520
1521

This specification was developed as a result of joint work of many individuals from the WSS TC including: TBD

1522
1523
1524
1525

The input specifications for this document were developed as a result of joint work with many individuals and teams, including: Keith Ballinger, Microsoft, Bob Blakley, IBM, Allen Brown, Microsoft, Joel Farrell, IBM, Mark Hayes, VeriSign, Kelvin Lawrence, IBM, Scott Konersmann, Microsoft, David Melgar, IBM, Dan Simon, Microsoft, Wayne Vicknair, IBM.

16References

1526

- 1527 **[DIGSIG]** Informational RFC 2828, "[Internet Security Glossary](#)," May 2000.
- 1528 **[Kerberos]** J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," [RFC 1510](#), September 1993, <http://www.ietf.org/rfc/rfc1510.txt>.
- 1529
- 1530 **[KEYWORDS]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," [RFC 2119](#), Harvard University, March 1997
- 1531
- 1532 **[SHA -1]** FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce / National Institute of Standards and Technology. <http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>
- 1533
- 1534
- 1535 **[SOAP11]** W3C Note, "[SOAP: Simple Object Access Protocol 1.1](#)," 08 May 2000.
- 1536 **[SOAP12]** **W3C Working Draft**, "SOAP Version 1.2 Part 1: Messaging Framework", 26 June 2002
- 1537
- 1538 **[SOAP-SEC]** W3C Note, "[SOAP Security Extensions: Digital Signature](#)," 06 February 2001.
- 1539
- 1540 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," [RFC 2396](#), MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.
- 1541
- 1542
- 1543 **[WS-Security]** "[Web Services Security Language](#)", IBM, Microsoft, VeriSign, April 2002.
- 1544 "[WS-Security Addendum](#)", IBM, Microsoft, VeriSign, August 2002.
- 1545 "[WS-Security XML Tokens](#)", IBM, Microsoft, VeriSign, August 2002.
- 1546 **[XML-C14N]** W3C Recommendation, "[Canonical XML Version 1.0](#)," 15 March 2001
- 1547 **[XML-Encrypt]** W3C Working Draft, "[XML Encryption Syntax and Processing](#)," 04 March 2002.
- 1548
- 1549 **[XML-ns]** W3C Recommendation, "[Namespaces in XML](#)," 14 January 1999.
- 1550 **[XML-Schema]** W3C Recommendation, "[XML Schema Part 1: Structures](#)," 2 May 2001.
- 1551 W3C Recommendation, "[XML Schema Part 2: Datatypes](#)," 2 May 2001.
- 1552 **[XML Signature]** W3C Recommendation, "[XML Signature Syntax and Processing](#)," 12 February 2002.
- 1553
- 1554 **[X509]** S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified Certificates Profile,"
- 1555 <http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I>
- 1556
- 1557
- 1558 **[XPath]** W3C Recommendation, "[XML Path Language](#)", 16 November 1999
- 1559 **[WSS-SAML]** OASIS Working Draft 02, "Web Services Security SAML Token Binding, 23 September 2002
- 1560
- 1561 **[WSS-XrML]** OASIS Working Draft 01, "Web Services Security XrML Token Binding, 20 September 2002
- 1562

1571

Appendix A: Utility Elements and Attributes

1572
1573
1574
1575
1576

This specification defines several elements, attributes, and attribute groups which can be re-used by other specifications. This appendix provides an overview of these *utility* components. It should be noted that the detailed descriptions are provided in the specification and this appendix will reference these sections as well as calling out other aspects not documented in the specification.

1577

A.1. Identification Attribute

1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591

There are many situations where elements within [SOAP](#) messages need to be referenced. For example, when signing a SOAP message, selected elements are included in the signature. [XML Schema Part 2](#) provides several built-in data types that may be used for identifying and referencing elements, but their use requires that consumers of the SOAP message either to have or be able to obtain the schemas where the identity or reference mechanisms are defined. In some circumstances, for example, intermediaries, this can be problematic and not desirable. Consequently a mechanism is required for identifying and referencing elements, based on the SOAP foundation, which does not rely upon complete schema knowledge of the context in which an element is used. This functionality can be integrated into SOAP processors so that elements can be identified and referred to without dynamic schema discovery and processing. This specification specifies a namespace-qualified global attribute for identifying an element which can be applied to any element that either allows arbitrary attributes or specifically allows this attribute. This is a general purpose mechanism which can be re-used as needed. A detailed description can be found in [Section 4.0 ID References](#).

1592

A.2. Timestamp Elements

1593
1594
1595
1596
1597
1598
1599
1600
1601

The specification defines XML elements which may be used to express timestamp information such as creation, expiration, and receipt. While defined in the context of messages, these elements can be re-used wherever these sorts of time statements need to be made. The elements in this specification are defined and illustrated using time references in terms of the *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this type for interoperability. It is further RECOMMENDED that all references be in UTC time for increased interoperability. If, however, other time types are used, then the *ValueType* attribute MUST be specified to indicate the data type of the time format. The following table provides an overview of these elements:

Element	Description
<wsu:Created>	This element is used to indicate the creation time associated with the enclosing context.
<wsu:Expires>	This element is used to indicate the expiration time associated with the enclosing context.

<wsu:Received>	This element is used to indicate the receipt time reference associated with the enclosing context.
----------------	--

1602 A detailed description can be found in Section [10 Message Timestamp](#).

1603 **A.3. General Schema Types**

1604 The schema for the utility aspects of this specification also defines some general purpose
 1605 schema elements. While these elements are defined in this schema for use with this
 1606 specification, they are general purpose definitions that may be used by other specifications as
 1607 well.

1608 Specifically, the following schema elements are defined and can be re-used:

Schema Element	Description
wsu:commonAtts attribute group	This attribute group defines the common attributes recommended for elements. This includes the wsu:Id attribute as well as extensibility for other namespace qualified attributes.
wsu:AttributedDateTime type	This type extends the XML Schema dateTime type to include the common attributes.
wsu:AttributedURIType	This type extends the XML Schema dateTime type to include the common attributes.

1609

1610

Appendix B: SecurityTokenReference Model

1611

This appendix provides a non-normative overview of the usage and processing models for the `<wsse:SecurityTokenReference>` element.

1612

1613

There are several motivations for introducing the `<wsse:SecurityTokenReference>` element:

1614

1615

The XML Signature reference mechanisms are focused on "key" references rather than general token references.

1616

1617

The XML Signature reference mechanisms utilize a fairly closed schema which limits the extensibility that can be applied.

1618

1619

There are additional types of general reference mechanisms that are needed, but are not covered by XML Signature.

1620

1621

There are scenarios where a reference may occur outside of an XML Signature and the XML

1622

Signature schema is not appropriate or desired.

1623

The XML Signature references may include aspects (e.g. transforms) that may not apply to all

1624

references.

1625

1626

The following use cases drive the above motivations:

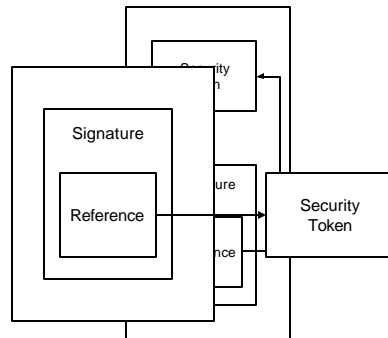
1627

Local Reference – A security token, that is included in the message in the `<wsse:Security>`

1628

header, is associated with an XML Signature. The figure below illustrates this:

1629



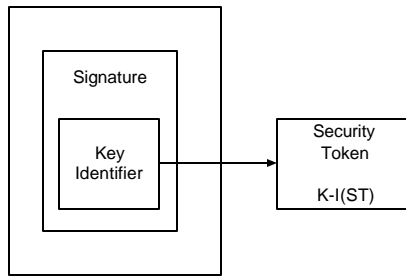
1630

Remote Reference – A security token, that is not included in the message but may be available at a specific URI, is associated with an XML Signature. The figure below illustrates this:

1631

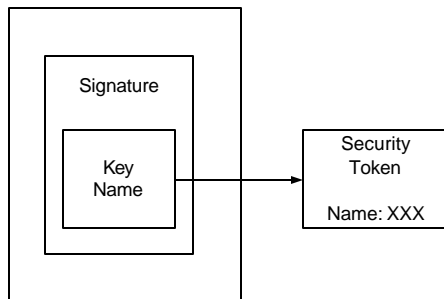
1632

1633 **Key Identifier** – A security token, which is associated with an XML Signature and identified using

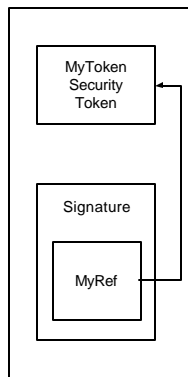


1634 a known value that is the result of a well-known function of the security token (defined by the
1635 token format or profile). The figure below illustrates this where the token is located externally:
1636

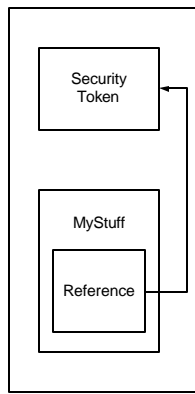
1637 **Key Name** – A security token is associated with an XML Signature and identified using a known
1638 value that represents a "name" assertion within the security token (defined by the token format or
1639 profile). The figure below illustrates this where the token is located externally:



1640 **Format-Specific References** – A security token is associated with an XML Signature and
1641 identified using a mechanism specific to the token (rather than the general mechanisms
1642



1643 described above). The figure below illustrates this:
1644



1645

1646 **Non-Signature References** – A message may contain XML that does not represent an XML
 1647 signature, but may reference a security token (which may or may not be included in the
 1648 message). The figure below illustrates this:

1649

1650

1651 All conformant implementations **MUST** be able to process the
 1652 `<wsse:SecurityTokenReference>` element. However, they are not required to support all of
 1653 the different types of references.

1654 The reference **MAY** include a *ValueType* attribute which provides a "hint" for the type of desired
 1655 token.

1656 If multiple sub-elements are specified, together they describe the reference for the token.

1657 There are several challenges that implementations face when trying to interoperate:

1658 **ID References** – The underlying XML referencing mechanism using the XML base type of ID
 1659 provides a simple straightforward XML element reference. However, because this is an XML
 1660 type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
 1661 requires the recipient to *understand* the schema. This may be an expensive task and in the
 1662 general case impossible as there is no way to know the "schema location" for a specific
 1663 namespace URI.

1664 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
 1665 references are, by definition, unique by XML. However, other mechanisms such as "principal
 1666 name" are not required to be unique and therefore such references may be unique.

1667 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
 1668 information about the "key" used in the signature. For token references within signatures, it is
 1669 RECOMMENDED that the `<wsse:SecurityTokenReference>` be placed within the
 1670 `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
 1671 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
 1672 Message Security or its profiles are preferred over the mechanisms in XML Signature.

Deleted: WS-Security

1673 The following provides additional details on the specific reference mechanisms defined in WSS:
 1674 SOAP Message Security:

Deleted: WS-Security

1675 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
 1676 the security token. If only the fragment is specified, then it references the security token within
 1677 the document whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to
 1678 a [potentially external] security token identified using a URI. There are no implied semantics
 1679 around the processing of the URI.

1680 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
 1681 by specifying a known value (identifier) for the token, which is determined by applying a special

1682 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
1683 specific security token but requires a profile or token-specific function to be specified. The
1684 *ValueType* attribute provide a *hint* as to the desired token type. The *EncodingType* attribute
1685 specifies how the unique value (identifier) is encoded. For example, a hash value may be
1686 encoded using base 64 encoding (the default).

1687 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
1688 specific value that is used to *match* identity assertion within the security token. This is a subset
1689 match and may result in multiple security tokens that match the specified name. While XML
1690 Signature doesn't imply formatting semantics, WSS: SOAP Message Security RECOMMENDS
1691 that X.509 names be specified.

Deleted: WS-Security

1692 It is expected that, where appropriate, profiles define if and how the reference mechanisms map
1693 to the specific token profile. Specifically, the profile should answer the following questions:

1694 What types of references can be used?
1695 How "Key Name" references map (if at all)?
1696 How "Key Identifier" references map (if at all)?
1697 Any additional profile or format-specific references?

1698

1699

Appendix C: Revision History

Rev	Date	What
01	20-Sep-02	Initial draft based on input documents and editorial review
02	24-Oct-02	Update with initial comments (technical and grammatical)
03	03-Nov-02	Feedback updates
04	17-Nov-02	Feedback updates
05	02-Dec-02	Feedback updates
06	08-Dec-02	Feedback updates
07	11-Dec-02	Updates from F2F
08	12-Dec-02	Updates from F2F

1702

Appendix D: Notices

1703 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1704 that might be claimed to pertain to the implementation or use of the technology described in this
1705 document or the extent to which any license under such rights might or might not be available;
1706 neither does it represent that it has made any effort to identify any such rights. Information on
1707 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1708 website. Copies of claims of rights made available for publication and any assurances of licenses
1709 to be made available, or the result of an attempt made to obtain a general license or permission
1710 for the use of such proprietary rights by implementors or users of this specification, can be
1711 obtained from the OASIS Executive Director.

1712 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1713 applications, or other proprietary rights which may cover technology that may be required to
1714 implement this specification. Please address the information to the OASIS Executive Director.

1715 Copyright © OASIS Open 2002. *All Rights Reserved.*

1716 This document and translations of it may be copied and furnished to others, and derivative works
1717 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1718 published and distributed, in whole or in part, without restriction of any kind, provided that the
1719 above copyright notice and this paragraph are included on all such copies and derivative works.
1720 However, this document itself does not be modified in any way, such as by removing the
1721 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1722 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1723 Property Rights document must be followed, or as required to translate it into languages other
1724 than English.

1725 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1726 successors or assigns.

1727 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1728 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1729 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1730 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1731 PARTICULAR PURPOSE.

1732