



1
2

3 **Web Services Security:**
4 **SOAP Message Security 1.1**
5 **(WS-Security 2004)**

6 **OASIS Public Review Draft - 28 June 2005**

7 **OASIS identifier:**
8 {product-productVersion-artifactType-stage-descriptiveName-revision.form (Word) (PDF)
9 (HTML)}

10 **Location:**
11 <http://docs.oasis-open.org/wss/2005/xx/wss-v1.1-spec-pr-SOAPMessageSecurity-01>

12 **Technical Committee:**
13 Web Service Security (WSS)

14 **Chairs:**
15 Kelvin Lawrence, IBM
16 Chris Kaler, Microsoft

17 **Editors:**
18 Anthony Nadalin, IBM
19 Chris Kaler, Microsoft
20 Ronald Monzillo, Sun
21 Phillip Hallam-Baker, Verisign

22 **Abstract:**
23 This specification describes enhancements to SOAP messaging to provide message
24 integrity and confidentiality. The specified mechanisms can be used to accommodate a
25 wide variety of security models and encryption technologies.

26
27 This specification also provides a general-purpose mechanism for associating security
28 tokens with message content. No specific type of security token is required, the
29 specification is designed to be extensible (i.e.. support multiple security token formats).
30 For example, a client might provide one format for proof of identity and provide another
31 format for proof that they have a particular business certification.

32
33 Additionally, this specification describes how to encode binary security tokens, a
34 framework for XML-based tokens, and how to include opaque encrypted keys. It also

35 includes extensibility mechanisms that can be used to further describe the characteristics
36 of the tokens that are included with a message.

37 **Status:**

38 This is a technical committee document submitted for consideration by the OASIS Web
39 Services Security (WSS) technical committee. Please send comments to the editors. If
40 you are on the wss@lists.oasis-open.org list for committee members, send comments
41 there. If you are not on that list, subscribe to the wss-comment@lists.oasis-open.org list
42 and send comments there. To subscribe, send an email message to [wss-comment-](mailto:wss-comment-request@lists.oasis-open.org)
43 [request@lists.oasis-open.org](mailto:wss-comment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message. For
44 patent disclosure information that may be essential to the implementation of this
45 specification, and any offers of licensing terms, refer to the Intellectual Property Rights
46 section of the OASIS Web Services Security Technical Committee (WSS TC) web page
47 at <http://www.oasis-open.org/committees/wss/ipr.php>. General OASIS IPR information
48 can be found at <http://www.oasis-open.org/who/intellectualproperty.shtml>.

50 Notices

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

60
61 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
62 applications, or other proprietary rights which may cover technology that may be required to
63 implement this specification. Please address the information to the OASIS Executive Director.
64 Copyright © OASIS Open 2002-2005. *All Rights Reserved.*

65
66 This document and translations of it may be copied and furnished to others, and derivative works
67 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
68 published and distributed, in whole or in part, without restriction of any kind, provided that the
69 above copyright notice and this paragraph are included on all such copies and derivative works.
70 However, this document itself does not be modified in any way, such as by removing the
71 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
72 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
73 Property Rights document must be followed, or as required to translate it into languages other
74 than English.

75
76 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
77 successors or assigns.

78
79 This document and the information contained herein is provided on an "AS IS" basis and OASIS
80 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
81 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
82 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
83 PARTICULAR PURPOSE.

84

85 **This section is non-normative.**

Table of Contents

87	1	Introduction	7
88	1.1	Goals and Requirements	7
89	1.1.1	Requirements.....	8
90	1.1.2	Non-Goals.....	8
91	2	Notations and Terminology.....	9
92	2.1	Notational Conventions	9
93	2.2	Namespaces	9
94	2.3	Acronyms and Abbreviations	10
95	2.4	Terminology.....	10
96	2.5	Note on Examples.....	12
97	3	Message Protection Mechanisms.....	13
98	3.1	Message Security Model.....	13
99	3.2	Message Protection.....	13
100	3.3	Invalid or Missing Claims	14
101	3.4	Example	14
102	4	ID References	16
103	4.1	Id Attribute.....	16
104	4.2	Id Schema	16
105	5	Security Header	18
106	6	Security Tokens	20
107	6.1	Attaching Security Tokens	20
108	6.1.1	Processing Rules	20
109	6.1.2	Subject Confirmation.....	20
110	6.2	User Name Token	20
111	6.2.1	Usernames.....	20
112	6.3	Binary Security Tokens	21
113	6.3.1	Attaching Security Tokens	21
114	6.3.2	Encoding Binary Security Tokens.....	21
115	6.4	XML Tokens	22
116	6.5	EncryptedData Token	22
117	6.6	Identifying and Referencing Security Tokens	23
118	7	Token References.....	24
119	7.1	SecurityTokenReference Element	24
120	7.2	Direct References.....	26
121	7.3	Key Identifiers.....	26
122	7.4	Embedded References	28
123	7.5	ds:KeyInfo	29
124	7.6	Key Names.....	29

125	7.7 Encrypted Key reference.....	29
126	8 Signatures.....	31
127	8.1 Algorithms.....	31
128	8.2 Signing Messages.....	33
129	8.3 Signing Tokens.....	34
130	8.4 Signature Validation.....	36
131	8.5 Signature Confirmation.....	37
132	8.5.1 Response Generation Rules.....	38
133	8.5.2 Response Processing Rules.....	38
134	8.6 Example.....	39
135	9 Encryption.....	40
136	9.1 xenc:ReferenceList.....	40
137	9.2 xenc:EncryptedKey.....	41
138	9.3 Encrypted Header.....	42
139	9.4 Processing Rules.....	42
140	9.4.1 Encryption.....	42
141	9.4.2 Decryption.....	43
142	9.4.3 Encryption with EncryptedHeader.....	43
143	9.4.4 Processing an EncryptedHeader.....	44
144	9.4.5 Processing the mustUnderstand attribute on EncryptedHeader.....	45
145	10 Security Timestamps.....	46
146	11 Extended Example.....	48
147	12 Error Handling.....	51
148	13 Security Considerations.....	52
149	13.1 General Considerations.....	52
150	13.2 Additional Considerations.....	52
151	13.2.1 Replay.....	52
152	13.2.2 Combining Security Mechanisms.....	53
153	13.2.3 Challenges.....	53
154	13.2.4 Protecting Security Tokens and Keys.....	53
155	13.2.5 Protecting Timestamps and Ids.....	54
156	13.2.6 Protecting against removal and modification of XML Elements.....	54
157	14 Interoperability Notes.....	56
158	15 Privacy Considerations.....	57
159	16 References.....	58
160	Appendix A: Acknowledgements.....	60
161	Appendix B: Revision History.....	62
162	Appendix C: Utility Elements and Attributes.....	63
163	16.1 Identification Attribute.....	63
164	16.2 Timestamp Elements.....	63
165	16.3 General Schema Types.....	64

166 Appendix D: SecurityTokenReference Model 65
167

168

1 Introduction

169 This OASIS specification is the result of significant new work by the WSS Technical Committee
170 and supersedes the input submissions, Web Service Security (WS-Security) Version 1.0 April 5,
171 2002 and Web Services Security Addendum Version 1.0 August 18, 2002.

172

173 This specification proposes a standard set of SOAP [SOAP11, SOAP12] extensions that can be
174 used when building secure Web services to implement message content integrity and
175 confidentiality. This specification refers to this set of extensions and modules as the “Web
176 Services Security: SOAP Message Security” or “WSS: SOAP Message Security”.

177

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

183

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

190

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

1.1 Goals and Requirements

194
195 The goal of this specification is to enable applications to conduct secure SOAP message
196 exchanges.

197

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

201

202 As with every security protocol, significant efforts must be applied to ensure that security
203 protocols constructed using this specification are not vulnerable to any one of a wide range of
204 attacks. The examples in this specification are meant to illustrate the syntax of these mechanisms
205 and are not intended as examples of combining these mechanisms in secure ways.

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

209

210 The requirements to support secure message exchange are listed below.

211 **1.1.1 Requirements**

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

- 214 • Multiple security token formats
- 215 • Multiple trust domains
- 216 • Multiple signature formats
- 217 • Multiple encryption technologies
- 218 • End-to-end message content security and not just transport-level security

219 **1.1.2 Non-Goals**

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

- 221
- 222 • Establishing a security context or authentication mechanisms.
- 223 • Key derivation.
- 224 • Advertisement and exchange of security policy.
- 225 • How trust is established or determined.
- 226 • Non-repudiation.
- 227

228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272

2 Notations and Terminology

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

2.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

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]).

When describing concrete XML schemas, this specification uses a convention where each member of an element's [children] or [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/>).

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

2.2 Namespaces

Namespace URIs (of the general form "some-URI") represents some application-dependent or context-dependent URI as defined in RFC 2396 [URI].

This specification is backwardly compatible with version 1.0. This means that URIs and schema elements defined in 1.0 remain unchanged and new schema elements and constants are defined using 1.1 namespaces and URIs.

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):

```
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-wssecurity-secext-1.1.xsd
```

This specification is designed to work with the general SOAP [SOAP11, SOAP12] message structure and message processing model, and should be applicable to any version of SOAP. The current SOAP 1.1 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.

The namespaces used in this document are shown in the following table (note that for brevity, the examples use the prefixes listed below but do not include the URIs – those listed below are assumed).

Prefix	Namespace
ds	http://www.w3.org/2000/09/xmldsig#
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsse11	http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-wssecurity-secext-1.1.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
xenc	http://www.w3.org/2001/04/xmlenc#

273
274
275
276
277
278
279

The URLs provided for the *wsse* and *wsu* namespaces can be used to obtain the schema files.

Most URI fragments defined in this document are relative to the following base URI unless otherwise stated:

<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0>

280 2.3 Acronyms and Abbreviations

281 The following (non-normative) table defines acronyms and abbreviations for this document.
282

Term	Definition
HMAC	Keyed-Hashing for Message Authentication
SHA-1	Secure Hash Algorithm 1
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
XML	Extensible Markup Language

283 2.4 Terminology

284 Defined below are the basic definitions for the security terminology used in this specification.
285

286 **Claim** – A *claim* is a declaration made by an entity (e.g. name, identity, key, group, privilege,
287 capability, etc).
288

289 **Claim Confirmation** – A *claim confirmation* is the process of verifying that a claim applies to
290 an entity.

291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324

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

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

Digital Signature – In this document, digital signature and signature are used interchangeably and have the same meaning.

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

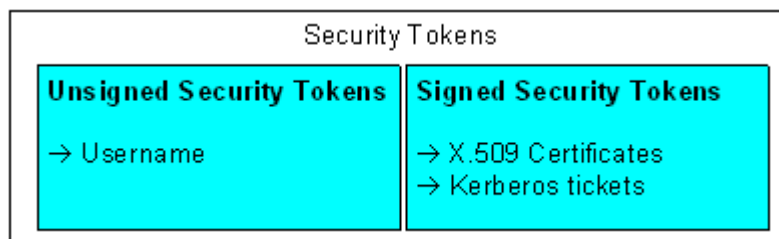
Integrity – *Integrity* is the property that data has not been modified.

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

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

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

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



325
326
327
328
329
330
331

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

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

332

2.5 Note on Examples

333

The examples which appear in this document are only intended to illustrate the correct syntax of the features being specified. The examples are NOT intended to necessarily represent best practice for implementing any particular security properties.

334

335

336

337

Specifically, the examples are constrained to contain only mechanisms defined in this document.

338

The only reason for this is to avoid requiring the reader to consult other documents merely to

339

understand the examples. It is NOT intended to suggest that the mechanisms illustrated

340

represent best practice or are the strongest available to implement the security properties in

341

question. In particular, mechanisms defined in other Token Profiles are known to be stronger,

342

more efficient and/or generally superior to some of the mechanisms shown in the examples in this

343

document.

344

3 Message Protection Mechanisms

345

346 When securing SOAP messages, various types of threats should be considered. This includes,
347 but is not limited to:

348

349

350

351

352

353

354

355

- the message could be modified or read by antagonists or
- an antagonist could send messages to a service that, while well-formed, lack appropriate security claims to warrant processing
- an antagonist could alter a message to the service which being well formed causes the service to process and respond to the client for an incorrect request.

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

356

3.1 Message Security Model

357

358

359

360

361

362

363

364

365

366

367

368

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

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 [X509] 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 producer of the containing message.

369

370

371

372

373

Signatures are used to verify message origin and integrity. Signatures are also used by message producers to demonstrate knowledge of the key, typically from a third party, 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.

374

375

376

377

378

379

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.

Where the specification requires that an element be "processed" it means that the element type MUST be recognized to the extent that an appropriate error is returned if the element is not supported.

380

3.2 Message Protection

381

382

383

384

385

386

387

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, or any combination of them (or parts of them).

Message integrity is provided by XML Signature [XMLSIG] in conjunction with security tokens to ensure that modifications to messages are detected. The integrity mechanisms are designed to

388 support multiple signatures, potentially by multiple SOAP actors/roles, and to be extensible to
389 support additional signature formats.
390
391 Message confidentiality leverages XML Encryption [XMLENC] in conjunction with security tokens
392 to keep portions of a SOAP message confidential. The encryption mechanisms are designed to
393 support additional encryption processes and operations by multiple SOAP actors/roles.
394
395 This document defines syntax and semantics of signatures within a <wsse:Security> element.
396 This document does not specify any signature appearing outside of a <wsse:Security>
397 element.

398 3.3 Invalid or Missing Claims

399 A message recipient SHOULD reject messages containing invalid signatures, messages missing
400 necessary claims or messages whose claims have unacceptable values. Such messages are
401 unauthorized (or malformed). This specification provides a flexible way for the message producer
402 to make a claim about the security properties by associating zero or more security tokens with the
403 message. An example of a security claim is the identity of the producer; the producer can claim
404 that he is Bob, known as an employee of some company, and therefore he has the right to send
405 the message.

406 3.4 Example

407 The following example illustrates the use of a custom security token and associated signature.
408 The token contains base64 encoded binary data conveying a symmetric key which, we assume,
409 can be properly authenticated by the recipient. The message producer uses the symmetric key
410 with an HMAC signing algorithm to sign the message. The message receiver uses its knowledge
411 of the shared secret to repeat the HMAC key calculation which it uses to validate the signature
412 and in the process confirm that the message was authored by the claimed user identity.

```
413  
414 (001) <?xml version="1.0" encoding="utf-8"?>  
415 (002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
416         xmlns:ds="...">  
417 (003)   <S11:Header>  
418 (004)     <wsse:Security  
419           xmlns:wsse="...">  
420 (005)       <wsse:BinarySecurityToken ValueType="  
421 http://fabrikam123#CustomToken "  
422         EncodingType="...#Base64Binary" wsu:Id=" MyID ">  
423 (006)         FHUIORv...  
424 (007)       </wsse:BinarySecurityToken>  
425 (008)       <ds:Signature>  
426 (009)         <ds:SignedInfo>  
427 (010)           <ds:CanonicalizationMethod  
428                 Algorithm=  
429                 "http://www.w3.org/2001/10/xml-exc-c14n#" />  
430 (011)           <ds:SignatureMethod  
431                 Algorithm=  
432                 "http://www.w3.org/2000/09/xmldsig#hmac-sha1" />  
433 (012)           <ds:Reference URI="#MsgBody">  
434 (013)             <ds:DigestMethod  
435                   Algorithm=  
436                   "http://www.w3.org/2000/09/xmldsig#sha1" />  
437 (014)             <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>  
438 (015)           </ds:Reference>
```

```

439      (016)          </ds:SignedInfo>
440      (017)          <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
441      (018)          <ds:KeyInfo>
442      (019)              <wsse:SecurityTokenReference>
443      (020)                  <wsse:Reference URI="#MyID" />
444      (021)              </wsse:SecurityTokenReference>
445      (022)          </ds:KeyInfo>
446      (023)          </ds:Signature>
447      (024)          </wsse:Security>
448      (025)      </S11:Header>
449      (026)      <S11:Body wsu:Id="MsgBody">
450      (027)          <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
451      (028)              QQQ
452      (029)          </tru:StockSymbol>
453      (028)      </S11:Body>
454      (029) </S11:Envelope>

```

455
456 The first two lines start the SOAP envelope. Line (003) begins the headers that are associated
457 with this SOAP message.

458
459 Line (004) starts the `<wsse:Security>` header defined in this specification. This header
460 contains security information for an intended recipient. This element continues until line (024).

461
462 Lines (005) to (007) specify a custom token that is associated with the message. In this case, it
463 uses an externally defined custom token format.

464
465 Lines (008) to (023) specify a digital signature. This signature ensures the integrity of the signed
466 elements. The signature uses the XML Signature specification identified by the ds namespace
467 declaration in Line (002).

468
469 Lines (009) to (016) describe what is being signed and the type of canonicalization being used.

470
471 Line (010) specifies how to canonicalize (normalize) the data that is being signed. Lines (012) to
472 (015) select the elements that are signed and how to digest them. Specifically, line (012)
473 indicates that the `<S11:Body>` element is signed. In this example only the message body is
474 signed; typically all critical elements of the message are included in the signature (see the
475 Extended Example below).

476
477 Line (017) specifies the signature value of the canonicalized form of the data that is being signed
478 as defined in the XML Signature specification.

479
480 Lines (018) to (022) provides information, partial or complete, as to where to find the security
481 token associated with this signature. Specifically, lines (019) to (021) indicate that the security
482 token can be found at (pulled from) the specified URL.

483
484 Lines (026) to (028) contain the body (payload) of the SOAP message.

485

486

4 ID References

487 There are many motivations for referencing other message elements such as signature
488 references or correlating signatures to security tokens. For this reason, this specification defines
489 the `wsu:Id` attribute so that recipients need not understand the full schema of the message for
490 processing of the security elements. That is, they need only "know" that the `wsu:Id` attribute
491 represents a schema type of ID which is used to reference elements. However, because some
492 key schemas used by this specification don't allow attribute extensibility (namely XML Signature
493 and XML Encryption), this specification also allows use of their local ID attributes in addition to
494 the `wsu:Id` attribute. As a consequence, when trying to locate an element referenced in a
495 signature, the following attributes are considered:

496

- 497 • Local ID attributes on XML Signature elements
- 498 • Local ID attributes on XML Encryption elements
- 499 • Global `wsu:Id` attributes (described below) on elements

500

501 In addition, when signing a part of an envelope such as the body, it is RECOMMENDED that an
502 ID reference is used instead of a more general transformation, especially XPath [XPath]. This is
503 to simplify processing.

504

4.1 Id Attribute

505 There are many situations where elements within SOAP messages need to be referenced. For
506 example, when signing a SOAP message, selected elements are included in the scope of the
507 signature. XML Schema Part 2 [XMLSCHEMA] provides several built-in data types that may be
508 used for identifying and referencing elements, but their use requires that consumers of the SOAP
509 message either have or must be able to obtain the schemas where the identity or reference
510 mechanisms are defined. In some circumstances, for example, intermediaries, this can be
511 problematic and not desirable.

512

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

517

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

521

4.2 Id Schema

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

525 The syntax for this attribute is as follows:

526

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

528

529 The following describes the attribute illustrated above:

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2005. All Rights Reserved.

28 June 2005
Page 16 of 68

530 .../@wsu:Id

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

533 Two `wsu:Id` attributes within an XML document MUST NOT have the same value.

534 Implementations MAY rely on XML Schema validation to provide rudimentary enforcement for
535 intra-document uniqueness. However, applications SHOULD NOT rely on schema validation
536 alone to enforce uniqueness.

537

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

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

541

```
<x:myElement wsu:Id="ID1" xmlns:x="..."  
xmlns:wsu="..." />
```

544

545 Conformance processors that do support XML Schema MUST treat this attribute as if it was
546 defined using a global attribute declaration.

547

548 Conformance processors that do not support dynamic XML Schema or DTDs discovery and
549 processing are strongly encouraged to integrate this attribute definition into their parsers. That is,
550 to treat this attribute information item as if its PSVI has a [type definition] which {target
551 namespace} is "`http://www.w3.org/2001/XMLSchema`" and which {type} is "ID." Doing so
552 allows the processor to inherently know *how* to process the attribute without having to locate and
553 process the associated schema. Specifically, implementations MAY support the value of the
554 `wsu:Id` as the valid identifier for use as an XPointer [XPointer] shorthand pointer for
555 interoperability with XML Signature references.

556

5 Security Header

557 The `<wsse:Security>` header block provides a mechanism for attaching security-related
558 information targeted at a specific recipient in the form of a SOAP actor/role. This may be either
559 the ultimate recipient of the message or an intermediary. Consequently, elements of this type
560 may be present multiple times in a SOAP message. An active intermediary on the message path
561 MAY add one or more new sub-elements to an existing `<wsse:Security>` header block if they
562 are targeted for its SOAP node or it MAY add one or more new headers for additional targets.
563

564 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted
565 for separate recipients. However, only one `<wsse:Security>` header block MAY omit the
566 `S11:actor` or `S12:role` attributes. Two `<wsse:Security>` header blocks MUST NOT have
567 the same value for `S11:actor` or `S12:role`. Message security information targeted for
568 different recipients MUST appear in different `<wsse:Security>` header blocks. This is due to
569 potential processing order issues (e.g. due to possible header re-ordering). The
570 `<wsse:Security>` header block without a specified `S11:actor` or `S12:role` MAY be
571 processed by anyone, but MUST NOT be removed prior to the final destination or endpoint.
572

573 As elements are added to a `<wsse:Security>` header block, they SHOULD be prepended to
574 the existing elements. As such, the `<wsse:Security>` header block represents the signing and
575 encryption steps the message producer took to create the message. This prepending rule
576 ensures that the receiving application can process sub-elements in the order they appear in the
577 `<wsse:Security>` header block, because there will be no forward dependency among the sub-
578 elements. Note that this specification does not impose any specific order of processing the sub-
579 elements. The receiving application can use whatever order is required.
580

581 When a sub-element refers to a key carried in another sub-element (for example, a signature
582 sub-element that refers to a binary security token sub-element that contains the X.509 certificate
583 used for the signature), the key-bearing element SHOULD be ordered to precede the key-using
584 Element:

585

```
586 <S11:Envelope>  
587   <S11:Header>  
588     ...  
589     <wsse:Security S11:actor="..." S11:mustUnderstand="...">  
590       ...  
591     </wsse:Security>  
592     ...  
593   </S11:Header>  
594   ...  
595 </S11:Envelope>
```

596

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

598 `/wsse:Security`

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

600

601 `/wsse:Security/@S11:actor`

602 This attribute allows a specific SOAP 1.1 [SPOAP11] actor to be identified. This attribute
603 is optional; however, no two instances of the header block may omit an actor or specify
604 the same actor.

605
606 */wsse:Security/@S12:role*

607 This attribute allows a specific SOAP 1.2 [SOAP12] role to be identified. This attribute is
608 optional; however, no two instances of the header block may omit a role or specify the
609 same role.

610
611 */wsse:Security/@S11:mustUnderstand*

612 This SOAP 1.1 [SPOAP11] attribute is used to indicate whether a header entry is
613 mandatory or optional for the recipient to process. The value of the mustUnderstand
614 attribute is either "1" or "0". The absence of the SOAP mustUnderstand attribute is
615 semantically equivalent to its presence with the value "0".

616
617 */wsse:Security/@S12:mustUnderstand*

618 This SOAP 1.2 [SPOAP12] attribute is used to indicate whether a header entry is
619 mandatory or optional for the recipient to process. The value of the mustUnderstand
620 attribute is either "true" or "false". The absence of the SOAP mustUnderstand attribute is
621 semantically equivalent to its presence with the value "false".

622
623 */wsse:Security/{any}*

624 This is an extensibility mechanism to allow different (extensible) types of security
625 information, based on a schema, to be passed. Unrecognized elements SHOULD cause
626 a fault.

627
628 */wsse:Security/@{any}*

629 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
630 added to the header. Unrecognized attributes SHOULD cause a fault.

631
632 All compliant implementations MUST be able to process a `<wsse:Security>` element.

633
634 All compliant implementations MUST declare which profiles they support and MUST be able to
635 process a `<wsse:Security>` element including any sub-elements which may be defined by that
636 profile. It is RECOMMENDED that undefined elements within the `<wsse:Security>` header
637 not be processed.

638
639 The next few sections outline elements that are expected to be used within a `<wsse:Security>`
640 header.

641
642 When a `<wsse:Security>` header includes a `mustUnderstand="true"` attribute:

- 643 • The receiver MUST generate a SOAP fault if does not implement the WSS: SOAP
644 Message Security specification corresponding to the namespace. Implementation means
645 ability to interpret the schema as well as follow the required processing rules specified in
646 WSS: SOAP Message Security.
- 647 • The receiver MUST generate a fault if unable to interpret or process security tokens
648 contained in the `<wsse:Security>` header block according to the corresponding WSS:
649 SOAP Message Security token profiles.
- 650 • Receivers MAY ignore elements or extensions within the `<wsse:Security>` element,
651 based on local security policy.

652

6 Security Tokens

653
654

This chapter specifies some different types of security tokens and how they are attached to messages.

655

6.1 Attaching Security Tokens

656
657
658
659

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

660
661

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

662

6.1.1 Processing Rules

663
664
665
666

This specification describes the processing rules for using and processing XML Signature and XML Encryption. These rules MUST be followed when using any type of security token. Note that if signature or encryption is used in conjunction with security tokens, they MUST be used in a way that conforms to the processing rules defined by this specification.

667

6.1.2 Subject Confirmation

668
669
670

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

671

6.2 User Name Token

672

6.2.1 Usernames

673
674
675

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

676
677
678
679
680

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

681
682

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

683

/wsse:UsernameToken

684

This element is used to represent a claimed identity.

685

686

/wsse:UsernameToken/@wsu:Id

687

A string label for this security token. The `wsu:Id` allow for an open attribute model.

688

689

/wsse:UsernameToken/wsse:Username

690

This required element specifies the claimed identity.

691
692 */wsse:UsernameToken/wsse:Username/@{any}*
693 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
694 added to the `<wsse:Username>` element.

695
696 */wsse:UsernameToken/{any}*
697 This is an extensibility mechanism to allow different (extensible) types of security
698 information, based on a schema, to be passed. Unrecognized elements SHOULD cause
699 a fault.

700
701 */wsse:UsernameToken/@{any}*
702 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
703 added to the `<wsse:UsernameToken>` element. Unrecognized attributes SHOULD
704 cause a fault.

705
706 All compliant implementations MUST be able to process a `<wsse:UsernameToken>` element.
707 The following illustrates the use of this:

```
708  
709 <S11:Envelope xmlns:S11="..." xmlns:wsse="...">  
710   <S11:Header>  
711     ...  
712     <wsse:Security>  
713       <wsse:UsernameToken>  
714         <wsse:Username>Zoe</wsse:Username>  
715       </wsse:UsernameToken>  
716     </wsse:Security>  
717     ...  
718   </S11:Header>  
719   ...  
720 </S11:Envelope>  
721
```

722 **6.3 Binary Security Tokens**

723 **6.3.1 Attaching Security Tokens**

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

727 **6.3.2 Encoding Binary Security Tokens**

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

732
733 The `<wsse:BinarySecurityToken>` element defines two attributes that are used to interpret
734 it. The `ValueType` attribute indicates what the security token is, for example, a Kerberos ticket.
735 The `EncodingType` tells how the security token is encoded, for example `Base64Binary`.
736 The following is an overview of the syntax:

737

```

738 <wsse:BinarySecurityToken wsu:Id=...
739                               EncodingType=...
740                               ValueType=.../>
741

```

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

/wsse:BinarySecurityToken

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

/wsse:BinarySecurityToken/@wsu:Id

An optional string label for this security token.

/wsse:BinarySecurityToken/@ValueType

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 URI that defines the value type and space of the encoded binary data. Subsequent specifications **MUST** define the `ValueType` value for the tokens that they define. The usage of `ValueType` is **RECOMMENDED**.

/wsse:BinarySecurityToken/@EncodingType

The `EncodingType` attribute is used to indicate, using a URI, the encoding format of the binary data (e.g., `base64` encoded). 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 (note that the URI fragments are relative to the URI for this specification):

URI	Description
#Base64Binary (default)	XML Schema base 64 encoding

/wsse:BinarySecurityToken/@{any}

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

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

6.4 XML Tokens

This section presents framework for using XML-based security tokens. Profile specifications describe rules and processes for specific XML-based security token formats.

6.5 EncryptedData Token

In certain cases it is desirable that the token included in the `<wsse:Security>` header be encrypted for the recipient processing role. In such a case the `<xenc:EncryptedData>` element **MAY** be used to contain a security token and included in the `<wsse:Security>` header. That is this specification defines the usage of `<xenc:EncryptedData>` to encrypt security tokens contained in `<wsse:Security>` header.

780
781 It should be noted that token references are not made to the `<xenc:EncryptedData>` element,
782 but instead to the token represented by the clear-text, once the `<xenc:EncryptedData>`
783 element has been processed (decrypted). Such references utilize the token profile for the
784 contained token. i.e., `<xenc:EncryptedData>` SHOULD NOT include an XML Id for
785 referencing the contained security token.
786
787 All `<xenc:EncryptedData>` tokens SHOULD either have an embedded encryption key or
788 should be referenced by a separate encryption key.
789 When a `<xenc:EncryptedData>` token is processed, it is replaced in the message infosect with
790 its decrypted form.

791 **6.6 Identifying and Referencing Security Tokens**

792 This specification also defines multiple mechanisms for identifying and referencing security
793 tokens using the `wsu:Id` attribute and the `<wsse:SecurityTokenReference>` element (as
794 well as some additional mechanisms). Please refer to the specific profile documents for the
795 appropriate reference mechanism. However, specific extensions MAY be made to the
796 `<wsse:SecurityTokenReference>` element.

797

7 Token References

798 This chapter discusses and defines mechanisms for referencing security tokens and other key
799 bearing elements..

800 7.1 SecurityTokenReference Element

801 Digital signature and encryption operations require that a key be specified. For various reasons,
802 the element containing the key in question may be located elsewhere in the message or
803 completely outside the message. The `<wsse:SecurityTokenReference>` element provides
804 an extensible mechanism for referencing security tokens and other key bearing elements.

805

806 The `<wsse:SecurityTokenReference>` element provides an open content model for
807 referencing key bearing elements because not all of them support a common reference pattern.
808 Similarly, some have closed schemas and define their own reference mechanisms. The open
809 content model allows appropriate reference mechanisms to be used.

810

811 If a `<wsse:SecurityTokenReference>` is used outside of the security header processing
812 block the meaning of the response and/or processing rules of the resulting references MUST be
813 specified by the containing element and are out of scope of this specification.

814 The following illustrates the syntax of this element:

815

```
816 <wsse:SecurityTokenReference wsu:Id="...">  
817   ...  
818 </wsse:SecurityTokenReference>
```

819

820 The following describes the elements defined above:

821

822 */wsse:SecurityTokenReference*

823 This element provides a reference to a security token.

824

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

826 A string label for this security token reference which names the reference. This attribute
827 does not indicate the ID of what is being referenced, that SHOULD be done using a
828 fragment URI in a `<wsse:Reference>` element within the
829 `<wsse:SecurityTokenReference>` element.

830

831 */wsse:SecurityTokenReference/@wsse:TokenType*

832 This optional attribute is used to identify, by URI, the type of the referenced token.
833 This specification recommends that token specific profiles define appropriate token type
834 identifying URI values, and that these same profiles require that these values be
835 specified in the profile defined reference forms.

836

837 When a `TokenType` attribute is specified in conjunction with a

838 `wsse:KeyIdentifier/@ValueType` attribute or a `wsse:Reference/@ValueType`

839 attribute that indicates the type of the referenced token, the security token type identified
840 by the TokenType attribute MUST be consistent with the security token type identified by
841 the ValueType attribute.
842

843 */wsse:SecurityTokenReference/@wsse:Usage*
844 This optional attribute is used to type the usage of the
845 `<wsse:SecurityTokenReference>`. Usages are specified using URIs and multiple
846 usages MAY be specified using XML list semantics. No usages are defined by this
847 specification.
848

849 */wsse:SecurityTokenReference/{any}*
850 This is an extensibility mechanism to allow different (extensible) types of security
851 references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
852 fault.
853

854 */wsse:SecurityTokenReference/@{any}*
855 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
856 added to the header. Unrecognized attributes SHOULD cause a fault.
857

858 All compliant implementations MUST be able to process a
859 `<wsse:SecurityTokenReference>` element.
860

861 This element can also be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to
862 retrieve the key information from a security token placed somewhere else. In particular, it is
863 RECOMMENDED, when using XML Signature and XML Encryption, that a
864 `<wsse:SecurityTokenReference>` element be placed inside a `<ds:KeyInfo>` to reference
865 the security token used for the signature or encryption.
866

867 There are several challenges that implementations face when trying to interoperate. Processing
868 the IDs and references requires the recipient to *understand* the schema. This may be an
869 expensive task and in the general case impossible as there is no way to know the "schema
870 location" for a specific namespace URI. As well, the primary goal of a reference is to uniquely
871 identify the desired token. ID references are, by definition, unique by XML. However, other
872 mechanisms such as "principal name" are not required to be unique and therefore such
873 references may be not unique.
874

875 The following list provides a list of the specific reference mechanisms defined in WSS: SOAP
876 Message Security in preferred order (i.e., most specific to least specific):
877

- 878 • **Direct References** – This allows references to included tokens using URI fragments and
879 external tokens using full URIs.
- 880 • **Key Identifiers** – This allows tokens to be referenced using an opaque value that
881 represents the token (defined by token type/profile).
- 882 • **Key Names** – This allows tokens to be referenced using a string that matches an identity
883 assertion within the security token. This is a subset match and may result in multiple
884 security tokens that match the specified name.
- 885 • **Embedded References** - This allows tokens to be embedded (as opposed to a pointer
886 to a token that resides elsewhere).

887 7.2 Direct References

888 The `<wsse:Reference>` element provides an extensible mechanism for directly referencing
889 security tokens using URIs.

890
891 The following illustrates the syntax of this element:

```
892 <wsse:SecurityTokenReference wsu:Id="...">  
893   <wsse:Reference URI="..." ValueType="..." />  
894 </wsse:SecurityTokenReference>
```

896
897 The following describes the elements defined above:

898
899 */wsse:SecurityTokenReference/wsse:Reference*

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

901

902 */wsse:SecurityTokenReference/wsse:Reference/@URI*

903 This optional attribute specifies an abstract URI for where to find a security token. If a
904 fragment is specified, then it indicates the local ID of the token being referenced.

905

906 */wsse:SecurityTokenReference/wsse:Reference/@ValueType*

907 This optional attribute specifies a URI that is used to identify the *type* of token being
908 referenced. This specification does not define any processing rules around the usage of
909 this attribute, however, specifications for individual token types MAY define specific
910 processing rules and semantics around the value of the URI and how it SHALL be
911 interpreted. If this attribute is not present, the URI MUST be processed as a normal URI.
912 The use of this attribute to identify the type of the referenced security token is
913 deprecated. Profiles which require or recommend the use of this attribute to identify the
914 type of the referenced security token SHOULD evolve to require or recommend the use
915 of the `wsse:SecurityTokenReference/@wsse:TokenType` attribute to identify the
916 type of the referenced token.

917

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

919 This is an extensibility mechanism to allow different (extensible) types of security
920 references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
921 fault.

922

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

924 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
925 added to the header. Unrecognized attributes SHOULD cause a fault.

926

927 The following illustrates the use of this element:

928

```
929 <wsse:SecurityTokenReference  
930   xmlns:wsse="...">  
931   <wsse:Reference  
932     URI="http://www.fabrikaml23.com/tokens/Zoe"/>  
933 </wsse:SecurityTokenReference>
```

934 7.3 Key Identifiers

935 Alternatively, if a direct reference is not used, then it is RECOMMENDED to use a key identifier to
936 specify/reference a security token instead of a `<ds:KeyName>`. A *bifier* is a value that can be
WSS: SOAP Message Security (WS-Security 2004) 28 June 2005

937 used to uniquely identify a security token (e.g. a hash of the important elements of the security
938 token). The exact value type and generation algorithm varies by security token type (and
939 sometimes by the data within the token), Consequently, the values and algorithms are described
940 in the token-specific profiles rather than this specification.

941
942 The <wsse:KeyIdentifier> element SHALL be placed in the
943 <wsse:SecurityTokenReference> element to reference a token using an identifier. This
944 element SHOULD be used for all key identifiers.

945
946 The processing model assumes that the key identifier for a security token is constant.
947 Consequently, processing a key identifier is simply looking for a security token whose key
948 identifier matches a given specified constant. The <wsse:KeyIdentifier> element is only
949 allowed inside a <wsse:SecurityTokenReference> element

950 The following is an overview of the syntax:

```
951 <wsse:SecurityTokenReference>  
952   <wsse:KeyIdentifier wsu:Id="..."  
953                       ValueType="..."  
954                       EncodingType="...">  
955     ...  
956   </wsse:KeyIdentifier>  
957 </wsse:SecurityTokenReference>
```

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

960
961 */wsse:SecurityTokenReference/wsse:KeyIdentifier*
962 This element is used to include a binary-encoded key identifier.

963
964 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@wsu:Id*
965 An optional string label for this identifier.

966
967 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@ValueType*
968 The optional `ValueType` attribute is used to indicate the type of `KeyIdentifier` being
969 used. This specification defines one `ValueType` that can be applied to all token types.
970 Each specific token profile specifies the `KeyIdentifier` types that may be used to
971 refer to tokens of that type. It also specifies the critical semantics of the identifier, such as
972 whether the `KeyIdentifier` is unique to the key or the token. If no value is specified
973 then the key identifier will be interpreted in an application-specific manner. This URI
974 fragment is relative to a base URI of
975 `http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-soap-`
976 `message-security-1.1`
977

978

URI	Description
<code>http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-soap-message-security-1.1#ThumbprintSHA1</code>	If the security token type that the Security Token Reference refers to already contains a representation for the thumbprint, the value obtained from the token MAY be used. If the token does not contain a representation of a thumbprint, then the value of the <code>KeyIdentifier</code> MUST be the SHA1 of the raw octets which

would be encoded within the security token element were it to be included.

979
980
981
982
983
984
985
986
987
988

/wsse:SecurityTokenReference/wsse:KeyIdentifier/@EncodingType

The optional `EncodingType` attribute is used to indicate, using a URI, the encoding format of the `KeyIdentifier` (`#Base64Binary`). This specification defines the `EncodingType` URI values appearing in the following table. A token specific profile MAY define additional token specific `EncodingType` URI values. A `KeyIdentifier` MUST include an `EncodingType` attribute when its `ValueType` is not sufficient to identify its encoding type. The base values defined in this specification are used (Note that URI fragments are relative to this document's URI):

URI	Description
<code>#Base64Binary</code>	XML Schema base 64 encoding

989
990
991
992

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

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

993 7.4 Embedded References

994 In some cases a reference may be to an embedded token (as opposed to a pointer to a token
995 that resides elsewhere). To do this, the `<wsse:Embedded>` element is specified within a
996 `<wsse:SecurityTokenReference>` element. The `<wsse:Embedded>` element is only
997 allowed inside a `<wsse:SecurityTokenReference>` element.

998 The following is an overview of the syntax:

999

```
1000 <wsse:SecurityTokenReference>  
1001   <wsse:Embedded wsu:Id="...">  
1002     ...  
1003   </wsse:Embedded>  
1004 </wsse:SecurityTokenReference>
```

1005

1006 The following describes the attributes and elements listed in the example above:

1007

1008 */wsse:SecurityTokenReference/wsse:Embedded*

This element is used to embed a token directly within a reference (that is, to create a *local* or *literal* reference).

1010

1011

1012 */wsse:SecurityTokenReference/wsse:Embedded/@wsu:Id*

An optional string label for this element. This allows this embedded token to be referenced by a signature or encryption.

1014

1015

1016 */wsse:SecurityTokenReference/wsse:Embedded/{any}*

This is an extensibility mechanism to allow any security token, based on schemas, to be embedded. Unrecognized elements SHOULD cause a fault.

1017

1018

1019

1020

/wsse:SecurityTokenReference/wsse:Embedded/@{any}

1021 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
1022 added. Unrecognized attributes SHOULD cause a fault.

1023

1024 The following example illustrates embedding a SAML assertion:

1025

```
1026 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">  
1027   <S11:Header>  
1028     <wsse:Security>  
1029       ...  
1030       <wsse:SecurityTokenReference>  
1031         <wsse:Embedded wsu:Id="tok1">  
1032           <saml:Assertion xmlns:saml="...">  
1033             ...  
1034           </saml:Assertion>  
1035         </wsse:Embedded>  
1036       </wsse:SecurityTokenReference>  
1037     </wsse:Security>  
1038   </S11:Header>  
1039   ...  
1040 </S11:Envelope>
```

1042 7.5 ds:KeyInfo

1043 The <ds:KeyInfo> element (from XML Signature) can be used for carrying the key information
1044 and is allowed for different key types and for future extensibility. However, in this specification,
1045 the use of <wsse:BinarySecurityToken> is the RECOMMENDED mechanism to carry key
1046 material if the key type contains binary data. Please refer to the specific profile documents for the
1047 appropriate way to carry key material.

1048

1049 The following example illustrates use of this element to fetch a named key:

1050

```
1051 <ds:KeyInfo Id="..." xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
1052   <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>  
1053 </ds:KeyInfo>
```

1054 7.6 Key Names

1055 It is strongly RECOMMENDED to use <wsse:KeyIdentifier> elements. However, if key
1056 names are used, then it is strongly RECOMMENDED that <ds:KeyName> elements conform to
1057 the attribute names in section 2.3 of RFC 2253 (this is recommended by XML Signature for
1058 <ds:X509SubjectName>) for interoperability.

1059

1060 Additionally, e-mail addresses, SHOULD conform to RFC 822:

```
1061 EmailAddress=ckaler@microsoft.com
```

1062 7.7 Encrypted Key reference

1063 In certain cases, an <xenc:EncryptedKey> element MAY be used to carry key material
1064 encrypted for the recipient's key. This key material is henceforth referred to as EncryptedKey.

1065

1066 The EncryptedKey MAY be used to perform other cryptographic operations within the same
1067 message, such as signatures. The EncryptedKey MAY also be used for performing

1068 cryptographic operations in subsequent messages exchanged by the two parties. Two
1069 mechanisms are defined for referencing the EncryptedKey.
1070
1071 When referencing the EncryptedKey within the same message that contains the
1072 <xenc:EncryptedKey> element, the <ds:KeyInfo> element of the referencing construct
1073 MUST contain a <wsse:SecurityTokenReference>. The
1074 <wsse:SecurityTokenReference> element MUST contain a <wsse:Reference> element.
1075
1076 The URI attribute value of the <wsse:Reference> element MUST be set to the value of the ID
1077 attribute of the referenced <xenc:EncryptedKey> element that contains the EncryptedKey.
1078 When referencing the EncryptedKey in a message that does not contain the
1079 <xenc:EncryptedKey> element, the <ds:KeyInfo> element of the referencing construct
1080 MUST contain a <wsse:SecurityTokenReference>. The
1081 <wsse:SecurityTokenReference> element MUST contain a <wsse:KeyIdentifier>
1082 element. The EncodingType attribute SHOULD be set to #Base64Binary. Other encoding
1083 types MAY be specified if agreed on by all parties. The ValueType attribute MUST be set to
1084 http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-soap-message-
1085 security-1.1#EncryptedKey. The identifier for a <xenc:EncryptedKey> token is defined
1086 as the SHA1 of the raw (pre-base64 encoding) octets specified in the <xenc:CipherValue>
1087 element of the referenced <xenc:EncryptedKey> token. This value is encoded as indicated in
1088 the KeyIdentifier reference. The ValueType attribute MUST be set to
1089 http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-soap-message-
1090 security-1.1#EncryptedKeySHA1

8 Signatures

1091

1092 Message producers may want to enable message recipients to determine whether a message
1093 was altered in transit and to verify that the claims in a particular security token apply to the
1094 producer of the message.

1095

1096 Demonstrating knowledge of a confirmation key associated with a token key-claim confirms the
1097 accompanying token claims. Knowledge of a confirmation key may be demonstrated using that
1098 key to create an XML Signature, for example. The relying party acceptance of the claims may
1099 depend on its confidence in the token. Multiple tokens may contain a key-claim for a signature
1100 and may be referenced from the signature using a `<wsse:SecurityTokenReference>`. A
1101 key-claim may be an X.509 Certificate token, or a Kerberos service ticket token to give two
1102 examples.

1103

1104 Because of the mutability of some SOAP headers, producers SHOULD NOT use the *Enveloped*
1105 *Signature Transform* defined in XML Signature. Instead, messages SHOULD explicitly include
1106 the elements to be signed. Similarly, producers SHOULD NOT use the *Enveloping Signature*
1107 defined in XML Signature [XMLSIG].

1108

1109 This specification allows for multiple signatures and signature formats to be attached to a
1110 message, each referencing different, even overlapping, parts of the message. This is important
1111 for many distributed applications where messages flow through multiple processing stages. For
1112 example, a producer may submit an order that contains an orderID header. The producer signs
1113 the orderID header and the body of the request (the contents of the order). When this is received
1114 by the order processing sub-system, it may insert a shippingID into the header. The order sub-
1115 system would then sign, at a minimum, the orderID and the shippingID, and possibly the body as
1116 well. Then when this order is processed and shipped by the shipping department, a shippedInfo
1117 header might be appended. The shipping department would sign, at a minimum, the shippedInfo
1118 and the shippingID and possibly the body and forward the message to the billing department for
1119 processing. The billing department can verify the signatures and determine a valid chain of trust
1120 for the order, as well as who authorized each step in the process.

1121

1122 All compliant implementations MUST be able to support the XML Signature standard.

1123

8.1 Algorithms

1124 This specification builds on XML Signature and therefore has the same algorithm requirements as
1125 those specified in the XML Signature specification.

1126 The following table outlines additional algorithms that are strongly RECOMMENDED by this
1127 specification:

1128

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

1129

1130

1131

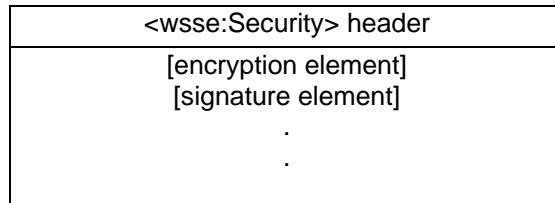
As well, the following table outlines additional algorithms that MAY be used:

Algorithm Type	Algorithm	Algorithm URI
Transform	SOAP Message Normalization	http://www.w3.org/TR/soap12-n11n/

1132
1133
1134
1135
1136
1137
1138
1139
1140
1141

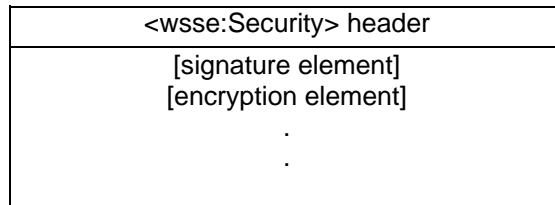
The Exclusive XML Canonicalization algorithm addresses the pitfalls of general canonicalization that can occur from *leaky* namespaces with pre-existing signatures.

Finally, if a producer wishes to sign a message before encryption, then following the ordering rules laid out in section 5, "Security Header", they SHOULD first prepend the signature element to the `<wsse:Security>` header, and then prepend the encryption element, resulting in a `<wsse:Security>` header that has the encryption element first, followed by the signature element:



1142
1143
1144
1145
1146
1147

Likewise, if a producer wishes to sign a message after encryption, they SHOULD first prepend the encryption element to the `<wsse:Security>` header, and then prepend the signature element. This will result in a `<wsse:Security>` header that has the signature element first, followed by the encryption element:



1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164

The XML Digital Signature WG has defined two canonicalization algorithms: XML Canonicalization and Exclusive XML Canonicalization. To prevent confusion, the first is also called Inclusive Canonicalization. Neither one solves all possible problems that can arise. The following informal discussion is intended to provide guidance on the choice of which one to use in particular circumstances. For a more detailed and technically precise discussion of these issues see: [XML-C14N] and [EXC-C14N].

There are two problems to be avoided. On the one hand, XML allows documents to be changed in various ways and still be considered equivalent. For example, duplicate namespace declarations can be removed or created. As a result, XML tools make these kinds of changes freely when processing XML. Therefore, it is vital that these equivalent forms match the same signature.

On the other hand, if the signature simply covers something like `xx:foo`, its meaning may change if `xx` is redefined. In this case the signature does not prevent tampering. It might be thought that the problem could be solved by expanding all the values in line. Unfortunately, there are

1165 mechanisms like XPATH which consider `xx="http://example.com/";` to be different from
1166 `yy="http://example.com/";` even though both `xx` and `yy` are bound to the same namespace.
1167 The fundamental difference between the Inclusive and Exclusive Canonicalization is the
1168 namespace declarations which are placed in the output. Inclusive Canonicalization copies all the
1169 declarations that are currently in force, even if they are defined outside of the scope of the
1170 signature. It also copies any `xml:` attributes that are in force, such as `xml:lang` or `xml:base`.
1171 This guarantees that all the declarations you might make use of will be unambiguously specified.
1172 The problem with this is that if the signed XML is moved into another XML document which has
1173 other declarations, the Inclusive Canonicalization will copy them and the signature will be invalid.
1174 This can even happen if you simply add an attribute in a different namespace to the surrounding
1175 context.
1176
1177 Exclusive Canonicalization tries to figure out what namespaces you are actually using and just
1178 copies those. Specifically, it copies the ones that are "visibly used", which means the ones that
1179 are a part of the XML syntax. However, it does not look into attribute values or element content,
1180 so the namespace declarations required to process these are not copied. For example
1181 if you had an attribute like `xx:foo="yy:bar"` it would copy the declaration for `xx`, but not `yy`. (This
1182 can even happen without your knowledge because XML processing tools will add `xsi:type` if
1183 you use a schema subtype.) It also does not copy the `xml:` attributes that are declared outside the
1184 scope of the signature.
1185
1186 Exclusive Canonicalization allows you to create a list of the namespaces that must be declared,
1187 so that it will pick up the declarations for the ones that are not visibly used. The only problem is
1188 that the software doing the signing must know what they are. In a typical SOAP software
1189 environment, the security code will typically be unaware of all the namespaces being used by the
1190 application in the message body that it is signing.
1191
1192 Exclusive Canonicalization is useful when you have a signed XML document that you wish to
1193 insert into other XML documents. A good example is a signed SAML assertion which might be
1194 inserted as a XML Token in the security header of various SOAP messages. The Issuer who
1195 signs the assertion will be aware of the namespaces being used and able to construct the list.
1196 The use of Exclusive Canonicalization will insure the signature verifies correctly every time.
1197 Inclusive Canonicalization is useful in the typical case of signing part or all of the SOAP body in
1198 accordance with this specification. This will insure all the declarations fall under the signature,
1199 even though the code is unaware of what namespaces are being used. At the same time, it is
1200 less likely that the signed data (and signature element) will be inserted in some other XML
1201 document. Even if this is desired, it still may not be feasible for other reasons, for example there
1202 may be Id's with the same value defined in both XML documents.
1203
1204 In other situations it will be necessary to study the requirements of the application and the
1205 detailed operation of the canonicalization methods to determine which is appropriate.
1206 This section is non-normative.

1207 **8.2 Signing Messages**

1208 The `<wsse:Security>` header block MAY be used to carry a signature compliant with the XML
1209 Signature specification within a SOAP Envelope for the purpose of signing one or more elements
1210 in the SOAP Envelope. Multiple signature entries MAY be added into a single SOAP Envelope
1211 within one `<wsse:Security>` header block. Producers SHOULD sign all important elements of
1212 the message, and careful thought must be given to creating a signing policy that requires signing
1213 of parts of the message that might legitimately be altered in transit.
1214

1215 SOAP applications MUST satisfy the following conditions:
1216

- 1217 • A compliant implementation MUST be capable of processing the required elements
1218 defined in the XML Signature specification.
- 1219 • To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element
1220 conforming to the XML Signature specification MUST be prepended to the existing
1221 content of the `<wsse:Security>` header block, in order to indicate to the receiver the
1222 correct order of operations. All the `<ds:Reference>` elements contained in the
1223 signature SHOULD refer to a resource within the enclosing SOAP envelope as described
1224 in the XML Signature specification. However, since the SOAP message exchange model
1225 allows intermediate applications to modify the Envelope (add or delete a header block; for
1226 example), XPath filtering does not always result in the same objects after message
1227 delivery. Care should be taken in using XPath filtering so that there is no subsequent
1228 validation failure due to such modifications.
- 1229 • The problem of modification by intermediaries (especially active ones) is applicable to
1230 more than just XPath processing. Digital signatures, because of canonicalization and
1231 digests, present particularly fragile examples of such relationships. If overall message
1232 processing is to remain robust, intermediaries must exercise care that the transformation
1233 algorithms used do not affect the validity of a digitally signed component.
- 1234 • Due to security concerns with namespaces, this specification strongly RECOMMENDS
1235 the use of the "Exclusive XML Canonicalization" algorithm or another canonicalization
1236 algorithm that provides equivalent or greater protection.
- 1237 • For processing efficiency it is RECOMMENDED to have the signature added and then
1238 the security token pre-pended so that a processor can read and cache the token before it
1239 is used.

1240 **8.3 Signing Tokens**

1241 It is often desirable to sign security tokens that are included in a message or even external to the
1242 message. The XML Signature specification provides several common ways for referencing
1243 information to be signed such as URIs, IDs, and XPath, but some token formats may not allow
1244 tokens to be referenced using URIs or IDs and XPaths may be undesirable in some situations.
1245 This specification allows different tokens to have their own unique reference mechanisms which
1246 are specified in their profile as extensions to the `<wsse:SecurityTokenReference>` element.
1247 This element provides a uniform referencing mechanism that is guaranteed to work with all token
1248 formats. Consequently, this specification defines a new reference option for XML Signature: the
1249 STR Dereference Transform.

1250
1251 This transform is specified by the URI `#STR-Transform` (Note that URI fragments are relative to
1252 this document's URI) and when applied to a `<wsse:SecurityTokenReference>` element it
1253 means that the output is the token referenced by the `<wsse:SecurityTokenReference>`
1254 element not the element itself.

1255
1256 As an overview the processing model is to echo the input to the transform except when a
1257 `<wsse:SecurityTokenReference>` element is encountered. When one is found, the element
1258 is not echoed, but instead, it is used to locate the token(s) matching the criteria and rules defined
1259 by the `<wsse:SecurityTokenReference>` element and echo it (them) to the output.
1260 Consequently, the output of the transformation is the resultant sequence representing the input
1261 with any `<wsse:SecurityTokenReference>` elements replaced by the referenced security
1262 token(s) matched.
1263

1264 The following illustrates an example of this transformation which references a token contained
1265 within the message envelope:

1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292

```
...  
<wsse:SecurityTokenReference wsu:Id="Str1">  
  ...  
</wsse:SecurityTokenReference>  
...  
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
  <ds:SignedInfo>  
    ...  
    <ds:Reference URI="#Str1">  
      <ds:Transforms>  
        <ds:Transform  
          Algorithm="...#STR-Transform">  
            <wsse:TransformationParameters>  
              <ds:CanonicalizationMethod  
                Algorithm="http://www.w3.org/TR/2001/REC-xml-  
c14n-20010315" />  
            </wsse:TransformationParameters>  
          </ds:Transform>  
          <ds:DigestMethod Algorithm=  
            "http://www.w3.org/2000/09/xmldsig#sha1" />  
          <ds:DigestValue>...</ds:DigestValue>  
        </ds:Reference>  
      </ds:SignedInfo>  
    <ds:SignatureValue></ds:SignatureValue>  
  </ds:Signature>  
...
```

1293
1294

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

1295

/wsse:TransformationParameters

1297
1298

This element is used to wrap parameters for a transformation allows elements even from the XML Signature namespace.

1299

/wsse:TransformationParameters/ds:Canonicalization

1300

This specifies the canolicalization algorithm to apply to the selected data.

1302

/wsse:TransformationParameters/{any}

1303

This is an extensibility mechanism to allow different (extensible) parameters to be specified in the future. Unrecognized parameters SHOULD cause a fault.

1305

/wsse:TransformationParameters/@{any}

1307

This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the element in the future. Unrecognized attributes SHOULD cause a fault.

1309

1310

1311

The following is a detailed specification of the transformation. The algorithm is identified by the URI: #STR-Transform.

1312

1313

1314

Transform Input:

1315

- The input is a node set. If the input is an octet stream, then it is automatically parsed; cf. XML Digital Signature [XMLSIG].

1316

1317

Transform Output:

1318

- The output is an octet steam.

- 1319 Syntax:
- 1320 • The transform takes a single mandatory parameter, a
- 1321 `<ds:CanonicalizationMethod>` element, which is used to serialize the input node
- 1322 set. Note, however, that the output may not be strictly in canonical form, per the
- 1323 canonicalization algorithm; however, the output is canonical, in the sense that it is
- 1324 unambiguous. However, because of syntax requirements in the XML Signature
- 1325 definition, this parameter **MUST** be wrapped in a
- 1326 `<wsse:TransformationParameters>` element.
- 1327 •

- 1328 Processing Rules:
- 1329 • Let N be the input node set.
- 1330 • Let R be the set of all `<wsse:SecurityTokenReference>` elements in N.
- 1331 • For each Ri in R, let Di be the result of dereferencing Ri.
- 1332 • If Di cannot be determined, then the transform **MUST** signal a failure.
- 1333 • If Di is an XML security token (e.g., a SAML assertion or a
- 1334 `<wsse:BinarySecurityToken>` element), then let Ri' be Di. Otherwise, Di is a raw
- 1335 binary security token; i.e., an octet stream. In this case, let Ri' be a node set consisting of
- 1336 a `<wsse:BinarySecurityToken>` element, utilizing the same namespace prefix as
- 1337 the `<wsse:SecurityTokenReference>` element Ri, with no `EncodingType` attribute,
- 1338 a `ValueType` attribute identifying the content of the security token, and text content
- 1339 consisting of the binary-encoded security token, with no white space.
- 1340 • Finally, employ the canonicalization method specified as a parameter to the transform to
- 1341 serialize N to produce the octet stream output of this transform; but, in place of any
- 1342 dereferenced `<wsse:SecurityTokenReference>` element Ri and its descendants,
- 1343 process the dereferenced node set Ri' instead. During this step, canonicalization of the
- 1344 replacement node set **MUST** be augmented as follows:
- 1345 ○ Note: A namespace declaration `xmlns=""` **MUST** be emitted with every apex
- 1346 element that has no namespace node declaring a value for the default
- 1347 namespace; cf. XML Decryption Transform.

1348

1349 Signing a `SecurityTokenReference` (STR) provides authentication and integrity protection

1350 of only the STR and not the referenced security token (ST). If signing the ST is the

1351 intended behavior, the STR Dereference Transform (STRDT) may be used which

1352 replaces the STR with the ST for digest computation, effectively protecting the ST and

1353 not the STR. If protecting both the ST and the STR is desired, you may sign the STR

1354 twice, once using the STRDT and once not using the STRDT.

1355

1356 The following table lists the full URI for each URI fragment referred to in the specification.

1357

URI Fragment	Full URI
#Base64Binary	http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-wss-soap-message-security-1.0#Base64Binary
#STR-Transform	http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-wss-soap-message-security-1.0#STR-Transform
#X509v3	http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-wss-x509-token-profile-1.0#X509v3

1358 8.4 Signature Validation

1359 The validation of a `<ds:Signature>` element inside an `<wsse:Security>` header block

1360 SHALL fail if:

WSS: SOAP Message Security (WS-Security 2004)
 Copyright © OASIS Open 2002-2005. All Rights Reserved.

28 June 2005
 Page 36 of 68

- 1361
- the syntax of the content of the element does not conform to this specification, or
 - the validation of the signature contained in the element fails according to the core validation of the XML Signature specification [XMLSIG], or
 - the application applying its own validation policy rejects the message for some reason (e.g., the signature is created by an untrusted key – verifying the previous two steps only performs cryptographic validation of the signature).
- 1362
- 1363
- 1364
- 1365
- 1366
- 1367

1368 If the validation of the signature element fails, applications MAY report the failure to the producer using the fault codes defined in Section 12 Error Handling.

1369

1370

1371 The signature validation shall additionally adhere to the rules defines in signature confirmation section below, if the initiator desires signature confirmation:

1372

1373 8.5 Signature Confirmation

1374 In the general model, the initiator uses XML Signature constructs to represent message parts of the request that were signed. The manifest of signed SOAP elements is contained in the

1375 <ds:Signature> element which in turn is placed inside the <wsse:Security> header. The

1376 <ds:Signature> element of the request contains a <ds:SignatureValue>. This element

1377 contains a base64 encoded value representing the actual digital signature. In certain situations it

1378 is desirable that initiator confirms that the message received was generated in response to a

1379 message it initiated in its unaltered form. This helps prevent certain forms of attack. This

1380 specification introduces a <wsse1:SignatureConfirmation> element to address this

1381 necessity.

1382

1383

1384 Compliant responder implementations that support signature confirmation, MUST include a

1385 <wsse1:SignatureConfirmation> element inside the <wsse:Security> header of the

1386 associated response message for every <ds:Signature> element that is a direct child of the

1387 <wsse:Security> header block in the originating message. The responder MUST include the

1388 contents of the <ds:SignatureValue> element of the request signature as the value of the

1389 @Value attribute of the <wsse1:SignatureConfirmation> element. The

1390 <wsse1:SignatureConfirmation> element MUST be included in the message signature of

1391 the associated response message.

1392

1393 If the associated originating signature is received in encrypted form then the corresponding

1394 <wsse1:SignatureConfirmation> element SHOULD be encrypted to protect the original

1395 signature and keys.

1396

1397 The schema outline for this element is as follows:

```
1398 <SignatureConfirmation wsu:Id="..." Value="..." />
```

1399 */SignatureConfirmation*

1400 This element indicates that the responder has processed the signature in the request.

1401 When this element is not present in a response the initiator SHOULD interpret that the

1402 responder is not compliant with this functionality.

1403

1404 */SignatureConfirmation/@wsu:Id*

1405 Identifier to be used when referencing this element in the SignedInfo reference list of the

1406 signature of the associated response message. This attribute MUST be present so that

1407 un-ambiguous references can be made to this <wsse1:SignatureConfirmation>

1408 element.

1409
1410 */SignatureConfirmation/@Value*
1411 This optional attribute contains the contents of a `<ds:SignatureValue>` copied from
1412 the associated request. If the request was not signed, then this attribute **MUST NOT** be
1413 present. If this attribute is specified with an empty value, the initiator **SHOULD** interpret
1414 this as incorrect behavior and process accordingly. When this attribute is not present, the
1415 initiator **SHOULD** interpret this to mean that the response is based on a request that was
1416 not signed.

1417 **8.5.1 Response Generation Rules**

1418 If the responder does not comply with this specification, it **MUST NOT** include any
1419 `<wssell:SignatureConfirmation>` elements in response messages it generates. If the
1420 responder complies with this specification, it **MUST** include at least one
1421 `<wssell:SignatureConfirmation>` element in the `<wsse:Security>` header in any
1422 response(s) associated with requests. That is, the normal messaging patterns are not altered.
1423 For every response message generated, the responder **MUST** include a
1424 `<wssell:SignatureConfirmation>` element for every `<ds:Signature>` element it
1425 processed from the original request message. The `Value` attribute **MUST** be set to the exact
1426 value of the `<ds:SignatureValue>` element of the corresponding `<ds:Signature>` element.
1427 If no `<ds:Signature>` elements are present in the original request message, the responder
1428 **MUST** include exactly one `<wssell:SignatureConfirmation>` element. The `Value` attribute
1429 of the `<wssell:SignatureConfirmation>` element **MUST NOT** be present. The responder
1430 **MUST** include all `<wssell:SignatureConfirmation>` elements in the message signature of
1431 the response message(s). If the `<ds:Signature>` element corresponding to a
1432 `<wssell:SignatureConfirmation>` element was encrypted in the original request message,
1433 the `<wssell:SignatureConfirmation>` element **SHOULD** be encrypted for the recipient of
1434 the response message(s).
1435

1436 **8.5.2 Response Processing Rules**

1437 The signature validation shall additionally adhere to the following processing guidelines, if the
1438 initiator desires signature confirmation:

- 1439 • If a response message does not contain a `<wssell:SignatureConfirmation>`
1440 element inside the `<wsse:Security>` header, the initiator **SHOULD** reject the response
1441 message.
- 1442 • If a response message does contain a `<wssell:SignatureConfirmation>` element
1443 inside the `<wsse:Security>` header but `@Value` attribute is not present on
1444 `<wssell:SignatureConfirmation>` element, and the associated request message
1445 did include a `<ds:Signature>` element, the initiator **SHOULD** reject the response
1446 message.
- 1447 • If a response message does contain a `<wssell:SignatureConfirmation>` element
1448 inside the `<wsse:Security>` header and the `@Value` attribute is present on the
1449 `<wssell:SignatureConfirmation>` element, but the associated request did not
1450 include a `<ds:Signature>` element, the initiator **SHOULD** reject the response
1451 message.
- 1452 • If a response message does contain a `<wssell:SignatureConfirmation>` element
1453 inside the `<wsse:Security>` header, and the associated request message did include
1454 a `<ds:Signature>` element and the `@Value` attribute is present but does not match the

1455 stored signature value of the associated request message, the initiator SHOULD reject
1456 the response message.
1457 • If a response message does not contain a <wssell:SignatureConfirmation>
1458 element inside the <wsse:Security> header corresponding to each
1459 <ds:Signature> element or if the @Value attribute present does not match the stored
1460 signature values of the associated request message, the initiator SHOULD reject the
1461 response message.

1462 8.6 Example

1463 The following sample message illustrates the use of integrity and security tokens. For this
1464 example, only the message body is signed.
1465

```
1466 <?xml version="1.0" encoding="utf-8"?>  
1467 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
1468 xmlns:ds="...">  
1469   <S11:Header>  
1470     <wsse:Security>  
1471       <wsse:BinarySecurityToken  
1472         ValueType="...#X509v3"  
1473         EncodingType="...#Base64Binary"  
1474         wsu:Id="X509Token">  
1475         MIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...  
1476       </wsse:BinarySecurityToken>  
1477       <ds:Signature>  
1478         <ds:SignedInfo>  
1479           <ds:CanonicalizationMethod Algorithm=  
1480             "http://www.w3.org/2001/10/xml-exc-c14n#" />  
1481           <ds:SignatureMethod Algorithm=  
1482             "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />  
1483           <ds:Reference URI="#myBody">  
1484             <ds:Transforms>  
1485               <ds:Transform Algorithm=  
1486                 "http://www.w3.org/2001/10/xml-exc-c14n#" />  
1487             </ds:Transforms>  
1488             <ds:DigestMethod Algorithm=  
1489               "http://www.w3.org/2000/09/xmldsig#sha1" />  
1490             <ds:DigestValue>EULddyts01...</ds:DigestValue>  
1491           </ds:Reference>  
1492         </ds:SignedInfo>  
1493         <ds:SignatureValue>  
1494         BL8jdfToEb11/vXcMZNNjPOV...  
1495       </ds:SignatureValue>  
1496       <ds:KeyInfo>  
1497         <wsse:SecurityTokenReference>  
1498           <wsse:Reference URI="#X509Token" />  
1499         </wsse:SecurityTokenReference>  
1500       </ds:KeyInfo>  
1501     </ds:Signature>  
1502   </wsse:Security>  
1503 </S11:Header>  
1504 <S11:Body wsu:Id="myBody">  
1505   <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">  
1506     QQQ  
1507   </tru:StockSymbol>  
1508 </S11:Body>  
1509 </S11:Envelope>
```

1510 9 Encryption

1511 This specification allows encryption of any combination of body blocks, header blocks, and any of
1512 these sub-structures by either a common symmetric key shared by the producer and the recipient
1513 or a symmetric key carried in the message in an encrypted form.

1514
1515 In order to allow this flexibility, this specification leverages the XML Encryption standard. This
1516 specification describes how the two elements `<xenc:ReferenceList>` and
1517 `<xenc:EncryptedKey>` listed below and defined in XML Encryption can be used within the
1518 `<wsse:Security>` header block. When a producer or an active intermediary encrypts
1519 portion(s) of a SOAP message using XML Encryption it MUST prepend a sub-element to the
1520 `<wsse:Security>` header block. Furthermore, the encrypting party MUST either prepend the
1521 sub-element to an existing `<wsse:Security>` header block for the intended recipients or create
1522 a new `<wsse:Security>` header block and insert the sub-element. The combined process of
1523 encrypting portion(s) of a message and adding one of these sub-elements is called an encryption
1524 step hereafter. The sub-element MUST contain the information necessary for the recipient to
1525 identify the portions of the message that it is able to decrypt.

1526
1527 This specification additionally defines an element `<wssell:EncryptedHeader>` for containing
1528 encrypted SOAP header blocks. This specification RECOMMENDS an additional mechanism that
1529 uses this element for encrypting SOAP header blocks that complies with SOAP processing
1530 guidelines while preserving the confidentiality of attributes on the SOAP header blocks.
1531 All compliant implementations MUST be able to support the XML Encryption standard [XMLENC].

1532 9.1 xenc:ReferenceList

1533 The `<xenc:ReferenceList>` element from XML Encryption [XMLENC] MAY be used to
1534 create a manifest of encrypted portion(s), which are expressed as `<xenc:EncryptedData>`
1535 elements within the envelope. An element or element content to be encrypted by this encryption
1536 step MUST be replaced by a corresponding `<xenc:EncryptedData>` according to XML
1537 Encryption. All the `<xenc:EncryptedData>` elements created by this encryption step
1538 SHOULD be listed in `<xenc:DataReference>` elements inside one or more
1539 `<xenc:ReferenceList>` element.

1540
1541 Although in XML Encryption [XMLENC], `<xenc:ReferenceList>` was originally designed to
1542 be used within an `<xenc:EncryptedKey>` element (which implies that all the referenced
1543 `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows
1544 that `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>`
1545 MAY be encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>`
1546 within individual `<xenc:EncryptedData>`.

1547
1548 A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the
1549 producer and the recipient use a shared secret key. The following illustrates the use of this sub-
1550 element:

```
1551  
1552     <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
1553     xmlns:ds="..." xmlns:xenc="...">  
1554     <S11:Header>
```

```

1555     <wsse:Security>
1556         <xenc:ReferenceList>
1557             <xenc:DataReference URI="#bodyID" />
1558         </xenc:ReferenceList>
1559     </wsse:Security>
1560 </S11:Header>
1561 <S11:Body>
1562     <xenc:EncryptedData Id="bodyID">
1563         <ds:KeyInfo>
1564             <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
1565         </ds:KeyInfo>
1566         <xenc:CipherData>
1567             <xenc:CipherValue>...</xenc:CipherValue>
1568         </xenc:CipherData>
1569     </xenc:EncryptedData>
1570 </S11:Body>
1571 </S11:Envelope>

```

9.2 xenc:EncryptedKey

1572
1573 When the encryption step involves encrypting elements or element contents within a SOAP
1574 envelope with a symmetric key, which is in turn to be encrypted by the recipient's key and
1575 embedded in the message, <xenc:EncryptedKey> MAY be used for carrying such an
1576 encrypted key. This sub-element SHOULD have a manifest, that is, an
1577 <xenc:ReferenceList> element, in order for the recipient to know the portions to be
1578 decrypted with this key. An element or element content to be encrypted by this encryption step
1579 MUST be replaced by a corresponding <xenc:EncryptedData> according to XML Encryption.
1580 All the <xenc:EncryptedData> elements created by this encryption step SHOULD be listed in
1581 the <xenc:ReferenceList> element inside this sub-element.

1582
1583 This construct is useful when encryption is done by a randomly generated symmetric key that is
1584 in turn encrypted by the recipient's public key. The following illustrates the use of this element:

```

1585 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1586 xmlns:ds="..." xmlns:xenc="...">
1587   <S11:Header>
1588     <wsse:Security>
1589       <xenc:EncryptedKey>
1590         ...
1591         <ds:KeyInfo>
1592           <wsse:SecurityTokenReference>
1593             <ds:X509IssuerSerial>
1594               <ds:X509IssuerName>
1595                 DC=ACMECorp, DC=com
1596               </ds:X509IssuerName>
1597             <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
1598           </ds:X509IssuerSerial>
1599           </wsse:SecurityTokenReference>
1600         </ds:KeyInfo>
1601         ...
1602       </xenc:EncryptedKey>
1603     </wsse:Security>
1604   </S11:Header>
1605   <S11:Body>
1606     <xenc:EncryptedData Id="bodyID">

```

```
1609         <xenc:CipherData>
1610             <xenc:CipherValue>...</xenc:CipherValue>
1611         </xenc:CipherData>
1612     </xenc:EncryptedData>
1613 </S11:Body>
1614 </S11:Envelope>
```

1615
1616 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in
1617 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that
1618 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

1619 **9.3 Encrypted Header**

1620 In order to be compliant with SOAP mustUnderstand processing guidelines and to prevent
1621 disclosure of information contained in attributes on a SOAP header block, this specification
1622 introduces an `<wsse11:EncryptedHeader>` element. This element contains exactly one
1623 `<xenc:EncryptedData>` element. This specification RECOMMENDS the use of
1624 `<wsse11:EncryptedHeader>` element for encrypting SOAP header blocks.

1625 **9.4 Processing Rules**

1626 Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the
1627 XML Encryption specification. An encrypted SOAP envelope MUST still be a valid SOAP
1628 envelope. The message creator MUST NOT encrypt the `<S11:Envelope>`,
1629 `<S12:Envelope>`, or `<S11:Body>`, `<S12:Body>` elements but MAY encrypt child elements of
1630 either the `<S11:Header>`, `<S12:Header>` and `<S11:Body>` or `<S12:Body>` elements.
1631 Multiple steps of encryption MAY be added into a single `<wsse:Security>` header block if they
1632 are targeted for the same recipient.

1633
1634 When an element or element content inside a SOAP envelope (e.g. the contents of the
1635 `<S11:Body>` or `<S12:Body>` elements) are to be encrypted, it MUST be replaced by an
1636 `<xenc:EncryptedData>`, according to XML Encryption and it SHOULD be referenced from the
1637 `<xenc:ReferenceList>` element created by this encryption step. If the target of reference is
1638 an EncryptedHeader as defined in section 9.3 above, see processing rules defined in section
1639 9.5.3 Encryption using EncryptedHeader and section 9.5.4 Decryption of EncryptedHeader
1640 below.

1641 **9.4.1 Encryption**

1642 The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1643 this specification are listed below (note that use of `<xenc:ReferenceList>` is
1644 RECOMMENDED. Additionally, if target of encryption is a SOAP header, processing rules
1645 defined in section 9.5.3 SHOULD be used).

- 1646 • Create a new SOAP envelope.
- 1647 • Create a `<wsse:Security>` header
- 1648 • When an `<xenc:EncryptedKey>` is used, create a `<xenc:EncryptedKey>` sub-
1649 element of the `<wsse:Security>` element. This `<xenc:EncryptedKey>` sub-
1650 element SHOULD contain an `<xenc:ReferenceList>` sub-element, containing a
1651 `<xenc:DataReference>` to each `<xenc:EncryptedData>` element that was
1652 encrypted using that key.

- 1653
- 1654
- 1655
- 1656
- 1657
- 1658
- 1659
- 1660
- 1661
- 1662
- 1663
- 1664
- 1665
- 1666
- Locate data items to be encrypted, i.e., XML elements, element contents within the target SOAP envelope.
 - Encrypt the data items as follows: For each XML element or element content within the target SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification [XMLENC]. Each selected original element or element content MUST be removed and replaced by the resulting `<xenc:EncryptedData>` element.
 - The optional `<ds:KeyInfo>` element in the `<xenc:EncryptedData>` element MAY reference another `<ds:KeyInfo>` element. Note that if the encryption is based on an attached security token, then a `<wsse:SecurityTokenReference>` element SHOULD be added to the `<ds:KeyInfo>` element to facilitate locating it.
 - Create an `<xenc:DataReference>` element referencing the generated `<xenc:EncryptedData>` elements. Add the created `<xenc:DataReference>` element to the `<xenc:ReferenceList>`.
 - Copy all non-encrypted data.

1667 9.4.2 Decryption

1668 On receiving a SOAP envelope containing encryption header elements, for each encryption
1669 header element the following general steps should be processed (this section is non-normative.
1670 Additionally, if the target of reference is an `EncryptedHeader`, processing rules as defined in
1671 section 9.5.4 below SHOULD be used):

- 1672
- 1673
- 1674
- 1675
- 1676
- 1677
- 1678
- 1679
- 1680
- 1681
- 1682
- 1683
- 1684
- 1685
- 1686
- 1687
- 1688
- 1689
- 1690
- 1691
1. Identify any decryption keys that are in the recipient's possession, then identifying any message elements that it is able to decrypt.
 2. Locate the `<xenc:EncryptedData>` items to be decrypted (possibly using the `<xenc:ReferenceList>`).
 3. Decrypt them as follows:
 - a. For each element in the target SOAP envelope, decrypt it according to the processing rules of the XML Encryption specification and the processing rules listed above.
 - b. If the decryption fails for some reason, applications MAY report the failure to the producer using the fault code defined in Section 12 Error Handling of this specification.
 - c. It is possible for overlapping portions of the SOAP message to be encrypted in such a way that they are intended to be decrypted by SOAP nodes acting in different Roles. In this case, the `<xenc:ReferenceList>` or `<xenc:EncryptedKey>` elements identifying these encryption operations will necessarily appear in different `<wsse:Security>` headers. Since SOAP does not provide any means of specifying the order in which different Roles will process their respective headers, this order is not specified by this specification and can only be determined by a prior agreement.

1692 9.4.3 Encryption with EncryptedHeader

1693 When it is required that an entire SOAP header block including the top-level element and its
1694 attributes be encrypted, the original header block SHOULD be replaced with a
1695 `<wsse11:EncryptedHeader>` element. The `<wsse11:EncryptedHeader>` element MUST
1696 contain the `<xenc:EncryptedData>` produced by encrypting the header block. A `wsu:Id`
1697 attribute MAY be added to the `<wsse11:EncryptedHeader>` element for referencing. If the
1698 referencing `<wsse:Security>` header block defines a value for the `<S12:mustUnderstand>`

1699 or <S11:mustUnderstand> attribute, that attribute and associated value MUST be copied to
1700 the <wssell:EncryptedHeader> element. If the referencing <wsse:Security> header
1701 block defines a value for the S12:Role or S11:Actor attribute, that attribute and associated value
1702 MUST be copied to the <wssell:EncryptedHeader> element.
1703

1704 Any header block can be replaced with a corresponding <wssell:EncryptedHeader> header
1705 block. This includes <wsse:Security> header blocks. (In this case, obviously if the encryption
1706 operation is specified in the same security header or in a security header targeted at a node
1707 which is reached after the node targeted by the <wssell:EncryptedHeader> element, the
1708 decryption will not occur.)

1709 In addition, <wssell:EncryptedHeader> header blocks can be super-encrypted and replaced
1710 by other <wssell:EncryptedHeader> header blocks (for wrapping/tunneling scenarios). Any
1711 <wsse:Security> header that encrypts a header block targeted to a particular actor SHOULD
1712 be targeted to that same actor, unless it is a security header.

1713 **9.4.4 Processing an EncryptedHeader**

1714 The processing model for <wssell:EncryptedHeader> header blocks is as follows:

- 1715 1. Resolve references to encrypted data specified in the <wsse:Security> header block
1716 targeted at this node. For each reference, perform the following steps.
- 1717 2. If the referenced element does not have a qualified name of
1718 <wssell:EncryptedHeader> then process as per section 9.5.2 Decryption and stop
1719 the processing steps here.
- 1720 3. Otherwise, extract the <xenc:EncryptedData> element from the
1721 <wssell:EncryptedHeader> element.
- 1722 4. Decrypt the contents of the <xenc:EncryptedData> element as per section 9.5.2
1723 Decryption and replace the <wssell:EncryptedHeader> element with the decrypted
1724 contents.
- 1725 5. Process the decrypted header block as per SOAP processing guidelines.

1726

1727 Alternatively, a processor may perform a pre-pass over the encryption references in the
1728 <wsse:Security> header:

- 1729 1. Resolve references to encrypted data specified in the <wsse:Security> header block
1730 targeted at this node. For each reference, perform the following steps.
- 1731 2. If a referenced element has a qualified name of <wssell:EncryptedHeader> then
1732 replace the <wssell:EncryptedHeader> element with the contained
1733 <xenc:EncryptedData> element and if present copy the value of the wsu:Id attribute
1734 from the <wssell:EncryptedHeader> element to the <xenc:EncryptedData>
1735 element.
- 1736 3. Process the <wsse:Security> header block as normal.

1737

1738 It should be noted that the results of decrypting a <wssell:EncryptedHeader> header block
1739 could be another <wssell:EncryptedHeader> header block. In addition, the result MAY be
1740 targeted at a different role than the role processing the <wssell:EncryptedHeader> header
1741 block.

1742 **9.4.5 Processing the mustUnderstand attribute on EncryptedHeader**

1743 If the `S11:mustUnderstand` or `S12:mustUnderstand` attribute is specified on the
1744 `<wsse11:EncryptedHeader>` header block, and is true, then the following steps define what it
1745 means to "understand" the `<wsse11:EncryptedHeader>` header block:

- 1746 1. The processor MUST be aware of this element and know how to decrypt and convert into
1747 the original header block. This DOES NOT REQUIRE that the process know that it has
1748 the correct keys or support the indicated algorithms.
- 1749 2. The processor MUST, after decrypting the encrypted header block, process the
1750 decrypted header block according to the SOAP processing guidelines. The receiver
1751 MUST raise a fault if any content required to adequately process the header block
1752 remains encrypted or if the decrypted SOAP header is not understood and the value of
1753 the `S12:mustUnderstand` or `S11:mustUnderstand` attribute on the decrypted
1754 header block is true. Note that in order to comply with SOAP processing rules in this
1755 case, the processor must roll back any persistent effects of processing the security
1756 header, such as storing a received token.
1757

10 Security Timestamps

1758

1759 It is often important for the recipient to be able to determine the *freshness* of security semantics.
1760 In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.
1761 This specification does not provide a mechanism for synchronizing time. The assumption is that
1762 time is trusted or additional mechanisms, not described here, are employed to prevent replay.
1763 This specification defines and illustrates time references in terms of the `xsd:dateTime` type
1764 defined in XML Schema. It is RECOMMENDED that all time references use this type. It is further
1765 RECOMMENDED that all references be in UTC time. Implementations MUST NOT generate time
1766 instants that specify leap seconds. If, however, other time types are used, then the `ValueType`
1767 attribute (described below) MUST be specified to indicate the data type of the time format.
1768 Requestors and receivers SHOULD NOT rely on other applications supporting time resolution
1769 finer than milliseconds.

1770

1771 The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and
1772 expiration times of the security semantics in a message.

1773

1774 All times MUST be in UTC format as specified by the XML Schema type (`dateTime`). It should be
1775 noted that times support time precision as defined in the XML Schema specification.

1776 The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and
1777 may only be present at most once per header (that is, per SOAP actor/role).

1778

1779 The ordering within the element is as illustrated below. The ordering of elements in the
1780 `<wsu:Timestamp>` element is fixed and MUST be preserved by intermediaries.

1781 The schema outline for the `<wsu:Timestamp>` element is as follows:

1782

```
1783 <wsu:Timestamp wsu:Id="...">  
1784   <wsu:Created ValueType="...">...</wsu:Created>  
1785   <wsu:Expires ValueType="...">...</wsu:Expires>  
1786   ...  
1787 </wsu:Timestamp>
```

1788

1789 The following describes the attributes and elements listed in the schema above:

1790

1791 */wsu:Timestamp*

1792 This is the element for indicating message timestamps.

1793

1794 */wsu:Timestamp/wsue:Created*

1795 This represents the creation time of the security semantics. This element is optional, but
1796 can only be specified once in a `<wsu:Timestamp>` element. Within the SOAP
1797 processing model, creation is the instant that the infoset is serialized for transmission.
1798 The creation time of the message SHOULD NOT differ substantially from its transmission
1799 time. The difference in time should be minimized.

1800

1801 */wsu:Timestamp/wsue:Expires*

1802 This element represents the expiration of the security semantics. This is optional, but
1803 can appear at most once in a `<wsu:Timestamp>` element. Upon expiration, the
1804 requestor asserts that its security semantics are no longer valid. It is strongly

1805 RECOMMENDED that recipients (anyone who processes this message) discard (ignore)
1806 any message whose security semantics have passed their expiration. A Fault code
1807 (`wsu:MessageExpired`) is provided if the recipient wants to inform the requestor that its
1808 security semantics were expired. A service MAY issue a Fault indicating the security
1809 semantics have expired.

1810
1811 */wsu:Timestamp/{any}*

1812 This is an extensibility mechanism to allow additional elements to be added to the
1813 element. Unrecognized elements SHOULD cause a fault.

1814

1815 */wsu:Timestamp/@wsu:Id*

1816 This optional attribute specifies an XML Schema ID that can be used to reference this
1817 element (the timestamp). This is used, for example, to reference the timestamp in a XML
1818 Signature.

1819

1820 */wsu:Timestamp/@{any}*

1821 This is an extensibility mechanism to allow additional attributes to be added to the
1822 element. Unrecognized attributes SHOULD cause a fault.

1823

1824 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
1825 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
1826 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
1827 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
1828 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
1829 judgment of the requestor's likely current clock time by means not described in this specification,
1830 for example an out-of-band clock synchronization protocol. The recipient may also use the
1831 creation time and the delays introduced by intermediate SOAP roles to estimate the degree of
1832 clock skew.

1833

1834 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

1835

```
1836 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">  
1837   <S11:Header>  
1838     <wsse:Security>  
1839       <wsu:Timestamp wsu:Id="timestamp">  
1840         <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>  
1841         <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>  
1842       </wsu:Timestamp>  
1843       ...  
1844     </wsse:Security>  
1845     ...  
1846   </S11:Header>  
1847   <S11:Body>  
1848     ...  
1849   </S11:Body>  
1850 </S11:Envelope>
```

1851

11 Extended Example

1852 The following sample message illustrates the use of security tokens, signatures, and encryption.
1853 For this example, the timestamp and the message body are signed prior to encryption. The
1854 decryption transformation is not needed as the signing/encryption order is specified within the
1855 <wsse:Security> header.

1856

```
1857 (001) <?xml version="1.0" encoding="utf-8"?>
1858 (002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1859 xmlns:xenc="..." xmlns:ds="...">
1860 (003)   <S11:Header>
1861 (004)     <wsse:Security>
1862 (005)       <wsu:Timestamp wsu:Id="T0">
1863 (006)         <wsu:Created>
1864 (007)           2001-09-13T08:42:00Z</wsu:Created>
1865 (008)         </wsu:Timestamp>
1866 (009)
1867 (010)       <wsse:BinarySecurityToken
1868             ValueType="...#X509v3"
1869             wsu:Id="X509Token"
1870             EncodingType="...#Base64Binary">
1871 (011) MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1872 (012) </wsse:BinarySecurityToken>
1873 (013)   <xenc:EncryptedKey>
1874 (014)     <xenc:EncryptionMethod Algorithm=
1875             "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
1876 (015)     <ds:KeyInfo>
1877             <wsse:SecurityTokenReference>
1878 (016)       <wsse:KeyIdentifier
1879             EncodingType="...#Base64Binary"
1880             ValueType="...#X509v3">MIGfMa0GCSq...
1881 (017)       </wsse:KeyIdentifier>
1882 (018)     </ds:KeyInfo>
1883 (019)     <xenc:CipherData>
1884 (020)       <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1885 (021)       </xenc:CipherValue>
1886 (022)     </xenc:CipherData>
1887 (023)     <xenc:ReferenceList>
1888 (024)       <xenc:DataReference URI="#encl1"/>
1889 (025)     </xenc:ReferenceList>
1890 (026)   </xenc:EncryptedKey>
1891 (027)   <ds:Signature>
1892 (028)     <ds:SignedInfo>
1893 (029)       <ds:CanonicalizationMethod
1894             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1895 (030)       <ds:SignatureMethod
1896             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
1897 (031)       <ds:Reference URI="#T0">
1898 (032)         <ds:Transforms>
1899 (033)           <ds:Transform
1900             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1901 (034)         </ds:Transforms>
1902 (035)         <ds:DigestMethod
1903             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```

1904 (036) <ds:DigestValue>LyLsF094hPi4wPU...
1905 (037) </ds:DigestValue>
1906 (038) </ds:Reference>
1907 (039) <ds:Reference URI="#body">
1908 (040) <ds:Transforms>
1909 (041) <ds:Transform
1910 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1911 (042) </ds:Transforms>
1912 (043) <ds:DigestMethod
1913 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1914 (044) <ds:DigestValue>LyLsF094hPi4wPU...
1915 (045) </ds:DigestValue>
1916 (046) </ds:Reference>
1917 (047) </ds:SignedInfo>
1918 (048) <ds:SignatureValue>
1919 (049) Hp1ZkmFZ/2kQLXDJbchm5gK...
1920 (050) </ds:SignatureValue>
1921 (051) <ds:KeyInfo>
1922 (052) <wsse:SecurityTokenReference>
1923 (053) <wsse:Reference URI="#X509Token" />
1924 (054) </wsse:SecurityTokenReference>
1925 (055) </ds:KeyInfo>
1926 (056) </ds:Signature>
1927 (057) </wsse:Security>
1928 (058) </S11:Header>
1929 (059) <S11:Body wsu:Id="body">
1930 (060) <xenc:EncryptedData
1931 Type="http://www.w3.org/2001/04/xmlenc#Element"
1932 wsu:Id="encl">
1933 (061) <xenc:EncryptionMethod
1934 Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-
1935 cbc" />
1936 (062) <xenc:CipherData>
1937 (063) <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1938 (064) </xenc:CipherValue>
1939 (065) </xenc:CipherData>
1940 (066) </xenc:EncryptedData>
1941 (067) </S11:Body>
1942 (068) </S11:Envelope>

```

1943

1944 Let's review some of the key sections of this example:

1945 Lines (003)-(058) contain the SOAP message headers.

1946

1947 Lines (004)-(057) represent the `<wsse:Security>` header block. This contains the security-
1948 related information for the message.

1949

1950 Lines (005)-(008) specify the timestamp information. In this case it indicates the creation time of
1951 the security semantics.

1952

1953 Lines (010)-(012) specify a security token that is associated with the message. In this case, it
1954 specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64
1955 encoding of the certificate.

1956

1957 Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a
1958 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
1959 encrypt the key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the

1960 symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines
1961 (023)-(025) identify the encryption block in the message that uses this symmetric key. In this
1962 case it is only used to encrypt the body (Id="enc1").
1963
1964 Lines (027)-(056) specify the digital signature. In this example, the signature is based on the
1965 X.509 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039)
1966 references the message body.
1967
1968 Lines (048)-(050) indicate the actual signature value – specified in Line (043).
1969
1970 Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the X.509
1971 certificate included in the message. Line (053) provides a URI link to the Lines (010)-(012).
1972 The body of the message is represented by Lines (059)-(067).
1973
1974 Lines (060)-(066) represent the encrypted metadata and form of the body using XML Encryption.
1975 Line (060) indicates that the "element value" is being replaced and identifies this encryption. Line
1976 (061) specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the
1977 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
1978 key as the key references this encryption – Line (024).
1979

1980

12 Error Handling

1981

There are many circumstances where an *error* can occur while processing security information.

1982

For example:

1983

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

1984

- Invalid or unauthenticated or unauthenticatable security token

1985

- Invalid signature

1986

- Decryption failure

1987

- Referenced security token is unavailable

1988

- Unsupported namespace

1989

1990

If a service does not perform its normal operation because of the contents of the Security header,

1991

then that MAY be reported using SOAP's Fault Mechanism. This specification does not mandate

1992

that faults be returned as this could be used as part of a denial of service or cryptographic

1993

attack. We combine signature and encryption failures to mitigate certain types of attacks.

1994

1995

If a failure is returned to a producer then the failure MUST be reported using the SOAP Fault

1996

mechanism. The following tables outline the predefined security fault codes. The "unsupported"

1997

classes of errors are as follows. Note that the reason text provided below is RECOMMENDED,

1998

but alternative text MAY be provided if more descriptive or preferred by the implementation. The

1999

tables below are defined in terms of SOAP 1.1. For SOAP 1.2, the Fault/Code/Value is

2000

env:Sender (as defined in SOAP 1.2) and the Fault/Code/Subcode/Value is the *faultcode* below

2001

and the Fault/Reason/Text is the *faultstring* below.

2002

Error that occurred (faultstring)	Faultcode
An unsupported token was provided	wsse:UnsupportedSecurityToken
An unsupported signature or encryption algorithm was used	wsse:UnsupportedAlgorithm

2003

2004

2005

The "failure" class of errors are:

Error that occurred (faultstring)	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

2006

13 Security Considerations

2007

2008

2009

2010

2011

2012

2013

As stated in the Goals and Requirements section of this document, this specification is meant to provide extensible framework and flexible syntax, with which one could implement various security mechanisms. This framework and syntax by itself *does not provide any guarantee of security*. When implementing and using this framework and syntax, one must make every effort to ensure that the result is not vulnerable to any one of a wide range of attacks.

2014

13.1 General Considerations

2015

2016

2017

2018

2019

It is not feasible to provide a comprehensive list of security considerations for such an extensible set of mechanisms. A complete security analysis **MUST** be conducted on specific solutions based on this specification. Below we illustrate some of the security concerns that often come up with protocols of this type, but we stress that this *is not an exhaustive list of concerns*.

2020

2021

2022

2023

2024

2025

2026

2027

2028

2029

2030

2031

2032

2033

2034

- freshness guarantee (e.g., the danger of replay, delayed messages and the danger of relying on timestamps assuming secure clock synchronization)
- proper use of digital signature and encryption (signing/encrypting critical parts of the message, interactions between signatures and encryption), i.e., signatures on (content of) encrypted messages leak information when in plain-text)
- protection of security tokens (integrity)
- certificate verification (including revocation issues)
- the danger of using passwords without outmost protection (i.e. dictionary attacks against passwords, replay, insecurity of password derived keys, ...)
- the use of randomness (or strong pseudo-randomness)
- interaction between the security mechanisms implementing this standard and other system component
- man-in-the-middle attacks
- PKI attacks (i.e. identity mix-ups)

2035

2036

2037

There are other security concerns that one may need to consider in security protocols. The list above should not be used as a "check list" instead of a comprehensive security analysis. The next section will give a few details on some of the considerations in this list.

2038

13.2 Additional Considerations

2039

13.2.1 Replay

2040

2041

2042

2043

2044

2045

2046

Digital signatures alone do not provide message authentication. One can record a signed message and resend it (a replay attack). It is strongly **RECOMMENDED** that messages include digitally signed elements to allow message recipients to detect replays of the message when the messages are exchanged via an open network. These can be part of the message or of the headers defined from other SOAP extensions. Four typical approaches are: Timestamp, Sequence Number, Expirations and Message Correlation. Signed timestamps **MAY** be used to keep track of messages (possibly by caching the most recent timestamp from a specific service)

2047 and detect replays of previous messages. It is RECOMMENDED that timestamps be cached for
2048 a given period of time, as a guideline, a value of five minutes can be used as a minimum to detect
2049 replays, and that timestamps older than that given period of time set be rejected in interactive
2050 scenarios.

2051 **13.2.2 Combining Security Mechanisms**

2052 This specification defines the use of XML Signature and XML Encryption in SOAP headers. As
2053 one of the building blocks for securing SOAP messages, it is intended to be used in conjunction
2054 with other security techniques. Digital signatures need to be understood in the context of other
2055 security mechanisms and possible threats to an entity.

2056
2057 Implementers should also be aware of all the security implications resulting from the use of digital
2058 signatures in general and XML Signature in particular. When building trust into an application
2059 based on a digital signature there are other technologies, such as certificate evaluation, that must
2060 be incorporated, but these are outside the scope of this document.

2061
2062 As described in XML Encryption, the combination of signing and encryption over a common data
2063 item may introduce some cryptographic vulnerability. For example, encrypting digitally signed
2064 data, while leaving the digital signature in the clear, may allow plain text guessing attacks.

2065 **13.2.3 Challenges**

2066 When digital signatures are used for verifying the claims pertaining to the sending entity, the
2067 producer must demonstrate knowledge of the confirmation key. One way to achieve this is to use
2068 a challenge-response type of protocol. Such a protocol is outside the scope of this document.
2069 To this end, the developers can attach timestamps, expirations, and sequences to messages.

2070 **13.2.4 Protecting Security Tokens and Keys**

2071 Implementers should be aware of the possibility of a token substitution attack. In any situation
2072 where a digital signature is verified by reference to a token provided in the message, which
2073 specifies the key, it may be possible for an unscrupulous producer to later claim that a different
2074 token, containing the same key, but different information was intended.

2075 An example of this would be a user who had multiple X.509 certificates issued relating to the
2076 same key pair but with different attributes, constraints or reliance limits. Note that the signature of
2077 the token by its issuing authority does not prevent this attack. Nor can an authority effectively
2078 prevent a different authority from issuing a token over the same key if the user can prove
2079 possession of the secret.

2080
2081 The most straightforward counter to this attack is to insist that the token (or its unique identifying
2082 data) be included under the signature of the producer. If the nature of the application is such that
2083 the contents of the token are irrelevant, assuming it has been issued by a trusted authority, this
2084 attack may be ignored. However because application semantics may change over time, best
2085 practice is to prevent this attack.

2086
2087 Requestors should use digital signatures to sign security tokens that do not include signatures (or
2088 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
2089 RECOMMENDED that all relevant and immutable message content be signed by the producer.
2090 Receivers SHOULD only consider those portions of the document that are covered by the
2091 producer's signature as being subject to the security tokens in the message. Security tokens
2092 appearing in `<wsse:Security>` header elements SHOULD be signed by their issuing authority

2093 so that message receivers can have confidence that the security tokens have not been forged or
2094 altered since their issuance. It is strongly RECOMMENDED that a message producer sign any
2095 <wsse:SecurityToken> elements that it is confirming and that are not signed by their issuing
2096 authority.
2097 When a requester provides, within the request, a Public Key to be used to encrypt the response,
2098 it is possible that an attacker in the middle may substitute a different Public Key, thus allowing the
2099 attacker to read the response. The best way to prevent this attack is to bind the encryption key in
2100 some way to the request. One simple way of doing this is to use the same key pair to sign the
2101 request as to encrypt the response. However, if policy requires the use of distinct key pairs for
2102 signing and encryption, then the Public Key provided in the request should be included under the
2103 signature of the request.

2104 **13.2.5 Protecting Timestamps and Ids**

2105 In order to *trust* wsu:Id attributes and <wsu:Timestamp> elements, they SHOULD be signed
2106 using the mechanisms outlined in this specification. This allows readers of the IDs and
2107 timestamps information to be certain that the IDs and timestamps haven't been forged or altered
2108 in any way. It is strongly RECOMMENDED that IDs and timestamp elements be signed.
2109

2110 **13.2.6 Protecting against removal and modification of XML Elements**

2111 XML Signatures using Shorthand XPointer References (AKA IDREF) protect against the removal
2112 and modification of XML elements; but do not protect the location of the element within the XML
2113 Document.

2114
2115 Whether or not this is security vulnerability depends on whether the location of the signed data
2116 within its surrounding context has any semantic import. This consideration applies to data carried
2117 in the SOAP Body or the Header.
2118

2119 Of particular concern is the ability to relocate signed data into a SOAP Header block which is
2120 unknown to the receiver and marked mustUnderstand="false". This could have the effect of
2121 causing the receiver to ignore signed data which the sender expected would either be processed
2122 or result in the generation of a mustUnderstand fault.
2123

2124 A similar exploit would involve relocating signed data into a SOAP Header block targeted to a
2125 S11:actor or S12:role other than that which the sender intended, and which the receiver will not
2126 process.
2127

2128 While these attacks could apply to any portion of the message, their effects are most pernicious
2129 with SOAP header elements which may not always be present, but must be processed whenever
2130 they appear.
2131

2132 In the general case of XML Documents and Signatures, this issue may be resolved by signing the
2133 entire XML Document and/or strict XML Schema specification and enforcement. However,
2134 because elements of the SOAP message, particularly header elements, may be legitimately
2135 modified by SOAP intermediaries, this approach is usually not appropriate. It is RECOMMENDED
2136 that applications signing any part of the SOAP body sign the entire body.
2137

2138 Alternatives countermeasures include (but are not limited to):

- 2139 • References using XPath transforms with Absolute Path expressions,

- 2140
- 2141
- 2142
- 2143
- 2144
- 2145
- 2146
- 2147
- 2148
- 2149
- 2150
- 2151
- 2152
- A Reference using an XPath transform to include any significant location-dependent elements and exclude any elements that might legitimately be removed, added, or altered by intermediaries,
 - Using only References to elements with location-independent semantics,
 - Strict policy specification and enforcement regarding which message parts are to be signed. For example:
 - Requiring that the entire SOAP Body and all children of SOAP Header be signed,
 - Requiring that SOAP header elements which are marked `mustUnderstand="false"` and have signed descendents **MUST** include the `mustUnderstand` attribute under the signature.
- This section is non-normative.

2153

14 Interoperability Notes

2154

Based on interoperability experiences with this and similar specifications, the following list highlights several common areas where interoperability issues have been discovered. Care should be taken when implementing to avoid these issues. It should be noted that some of these may seem "obvious", but have been problematic during testing.

2155

2156

2157

2158

2159

- **Key Identifiers:** Make sure you understand the algorithm and how it is applied to security tokens.

2160

2161

- **EncryptedKey:** The `<xenc:EncryptedKey>` element from XML Encryption requires a Type attribute whose value is one of a pre-defined list of values. Ensure that a correct value is used.

2162

2163

2164

- **Encryption Padding:** The XML Encryption random block cipher padding has caused issues with certain decryption implementations; be careful to follow the specifications exactly.

2165

2166

2167

- **IDs:** The specification recognizes three specific ID elements: the global `wsu:Id` attribute and the local `Id` attributes on XML Signature and XML Encryption elements (because the latter two do not allow global attributes). If any other element does not allow global attributes, it cannot be directly signed using an ID reference. Note that the global attribute `wsu:Id` MUST carry the namespace specification.

2168

2169

2170

2171

- **Time Formats:** This specification uses a restricted version of the XML Schema `xsd:dateTime` element. Take care to ensure compliance with the specified restrictions.

2172

2173

2174

- **Byte Order Marker (BOM):** Some implementations have problems processing the BOM marker. It is suggested that usage of this be optional.

2175

2176

- **SOAP, WSDL, HTTP:** Various interoperability issues have been seen with incorrect SOAP, WSDL, and HTTP semantics being applied. Care should be taken to carefully adhere to these specifications and any interoperability guidelines that are available.

2177

2178

2179

2180

This section is non-normative.

2181

15 Privacy Considerations

2182 In the context of this specification, we are only concerned with potential privacy violation by the
2183 security elements defined here. Privacy of the content of the payload message is out of scope.
2184 Producers or sending applications should be aware that claims, as collected in security tokens,
2185 are typically personal information, and should thus only be sent according to the producer's
2186 privacy policies. Future standards may allow privacy obligations or restrictions to be added to this
2187 data. Unless such standards are used, the producer must ensure by out-of-band means that the
2188 recipient is bound to adhering to all restrictions associated with the data, and the recipient must
2189 similarly ensure by out-of-band means that it has the necessary consent for its intended
2190 processing of the data.

2191

2192 If claim data are visible to intermediaries, then the policies must also allow the release to these
2193 intermediaries. As most personal information cannot be released to arbitrary parties, this will
2194 typically require that the actors are referenced in an identifiable way; such identifiable references
2195 are also typically needed to obtain appropriate encryption keys for the intermediaries.

2196 If intermediaries add claims, they should be guided by their privacy policies just like the original
2197 producers.

2198

2199 Intermediaries may also gain traffic information from a SOAP message exchange, e.g., who
2200 communicates with whom at what time. Producers that use intermediaries should verify that
2201 releasing this traffic information to the chosen intermediaries conforms to their privacy policies.

2202

2203 This section is non-normative.

2204

16References

- 2205 **[GLOSS]** Informational RFC 2828, "Internet Security Glossary," May 2000.
- 2206 **[KERBEROS]** J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993, <http://www.ietf.org/rfc/rfc1510.txt> .
- 2207
- 2208 **[KEYWORDS]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997
- 2209
- 2210 **[SHA-1]** FIPS PUB 180-1. Secure Hash Standard. U.S. Department of
2211 Commerce / National Institute of Standards and Technology.
2212 <http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>
- 2213 **[SOAP11]** W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.
- 2214 **[SOAP12]** W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging
2215 Framework", 23 June 2003
- 2216 **[SOAPSEC]** W3C Note, "SOAP Security Extensions: Digital Signature," 06 February
2217 2001.
- 2218 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers
2219 (URI): Generic Syntax," RFC 3986, MIT/LCS, Day Software, Adobe
2220 Systems, January 2005.
- 2221 **[XPath]** W3C Recommendation, "XML Path Language", 16 November 1999
- 2222
- 2223 The following are non-normative references included for background and related material:
- 2224 **[WS-SECURITY]** "Web Services Security Language", IBM, Microsoft, VeriSign, April 2002.
2225 "WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002.
2226 "WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002.
- 2227 **[XMLC14N]** W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001
- 2228 **[EXCC14N]** W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8
2229 July 2002.
- 2230 **[XMLENC]** W3C Working Draft, "XML Encryption Syntax and Processing," 04 March
2231 2002
- 2232 W3C Recommendation, "Decryption Transform for XML Signature", 10
2233 December 2002.
- 2234 **[XML-ns]** W3C Recommendation, "Namespaces in XML," 14 January 1999.
- 2235 **[XMLSCHEMA]** W3C Recommendation, "XML Schema Part 1: Structures," 2 May 2001.
2236 W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.
- 2237 **[XMLSIG]** D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-
2238 Signature Syntax and Processing*, W3C Recommendation, 12 February
2239 2002. <http://www.w3.org/TR/xmlsig-core/>.
- 2240 **[X509]** S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified
2241 Certificates Profile,"

2242		http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-l
2243		
2244	[WSS-SAML]	OASIS Working Draft 06, "Web Services Security SAML Token Profile", 21 February 2003
2245		
2246	[WSS-XrML]	OASIS Working Draft 03, "Web Services Security XrML Token Profile", 30 January 2003
2247		
2248	[WSS-X509]	OASIS, "Web Services Security X.509 Certificate Token Profile", 19 January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0
2249		
2250		
2251	[WSSKERBEROS]	OASIS Working Draft 03, "Web Services Security Kerberos Profile", 30 January 2003
2252		
2253	[WSSUSERNAME]	OASIS, "Web Services Security UsernameToken Profile" 19 January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
2254		
2255		
2256	[WSS-XCBF]	OASIS Working Draft 1.1, "Web Services Security XCBF Token Profile", 30 March 2003
2257		
2258	[XPOINTER]	"XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation", DeRose, Maler, Daniel, 11 September 2001.
2259		

Appendix A: Acknowledgements

Gene	Thurston	AmberPoint
Frank	Siebenlist	Argonne National Lab
Merlin	Hughes	Baltimore Technologies
Irving	Reid	Baltimore Technologies
Peter	Dapkus	BEA
Hal	Lockhart	BEA
Steve	Anderson	BMC (Sec)
Srinivas	Davanum	Computer Associates
Thomas	DeMartini	ContentGuard
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard
Shawn	Sharp	Cyclone Commerce
Ganesh	Vaideeswaran	Documentum
Sam	Wei	Documentum
John	Hughes	Entegrity
Tim	Moses	Entrust
Toshihiro	Nishimura	Fujitsu
Tom	Rutt	Fujitsu
Yutaka	Kudo	Hitachi
Jason	Rouault	HP
Paula	Austel	IBM
Bob	Blakley	IBM
Joel	Farrell	IBM
Satoshi	Hada	IBM
Maryann	Hondo	IBM
Michael	McIntosh	IBM
Hiroshi	Maruyama	IBM
David	Melgar	IBM
Anthony	Nadalin	IBM
Nataraj	Nagaratnam	IBM
Wayne	Vicknair	IBM
Kelvin	Lawrence	IBM (co-Chair)
Don	Flinn	Individual
Bob	Morgan	Individual
Bob	Atkinson	Microsoft
Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Paul	Cotton	Microsoft
Giovanni	Della-Libera	Microsoft
Vijay	Gajjala	Microsoft
Johannes	Klein	Microsoft
Scott	Konersmann	Microsoft
Chris	Kurt	Microsoft
Brian	LaMacchia	Microsoft

Paul	Leach	Microsoft
John	Manferdelli	Microsoft
John	Shewchuk	Microsoft
Dan	Simon	Microsoft
Hervey	Wilson	Microsoft
Chris	Kaler	Microsoft (co-Chair)
Prateek	Mishra	Netegrity
Frederick	Hirsch	Nokia
Senthil	Sengodan	Nokia
Lloyd	Burch	Novell
Ed	Reed	Novell
Charles	Knouse	Oblix
Vipin	Samar	Oracle
Jerry	Schwarz	Oracle
Eric	Gravengaard	Reactivity
Stuart	King	Reed Elsevier
Andrew	Nash	RSA Security
Rob	Philpott	RSA Security
Peter	Rostin	RSA Security
Martijn	de Boer	SAP
Blake	Dournaee	Sarvega
Pete	Wenzel	SeeBeyond
Jonathan	Tourzan	Sony
Yassir	Elley	Sun Microsystems
Jeff	Hodges	Sun Microsystems
Ronald	Monzillo	Sun Microsystems
Jan	Alexander	Systinet
Michael	Nguyen	The IDA of Singapore
Don	Adams	TIBCO
Symon	Chang	TIBCO
John	Weiland	US Navy
Phillip	Hallam-Baker	VeriSign
Mark	Hays	Verisign
Hemma	Prafullchandra	VeriSign

2261

Appendix B: Revision History

Rev	Date	By Whom	What
WGD 1.1	2004-09-13	Anthony Nadalin	Initial version cloned from the Version 1.1 and Errata
WGD 1.1	2005-02-14	Anthony Nadalin	Issues 250, 351, 352
WGD 1.1	2005-03-22	Anthony Nadalin	Issues 310, 373, 374
WGD 1.1	2005-05-11	Anthony Nadalin	Issues 390, 84
WGD 1.1	2005-05-17	Anthony Nadalin	Formatting Issues
WGD 1.1	2005-06-14	Anthony Nadalin	Issues 400, mustUnderstand

2263

2264

This section is non-normative.

2265

Appendix C: Utility Elements and Attributes

2266 These specifications define several elements, attributes, and attribute groups which can be re-
2267 used by other specifications. This appendix provides an overview of these *utility* components. It
2268 should be noted that the detailed descriptions are provided in the specification and this appendix
2269 will reference these sections as well as calling out other aspects not documented in the
2270 specification.

2271

16.1 Identification Attribute

2272 There are many situations where elements within SOAP messages need to be referenced. For
2273 example, when signing a SOAP message, selected elements are included in the signature. XML
2274 Schema Part 2 provides several built-in data types that may be used for identifying and
2275 referencing elements, but their use requires that consumers of the SOAP message either have or
2276 are able to obtain the schemas where the identity or reference mechanisms are defined. In some
2277 circumstances, for example, intermediaries, this can be problematic and not desirable.

2278

2279 Consequently a mechanism is required for identifying and referencing elements, based on the
2280 SOAP foundation, which does not rely upon complete schema knowledge of the context in which
2281 an element is used. This functionality can be integrated into SOAP processors so that elements
2282 can be identified and referred to without dynamic schema discovery and processing.

2283

2284 This specification specifies a namespace-qualified global attribute for identifying an element
2285 which can be applied to any element that either allows arbitrary attributes or specifically allows
2286 this attribute. This is a general purpose mechanism which can be re-used as needed.

2287

A detailed description can be found in Section 4.0 ID References.

2288

2289 This section is non-normative.

2290

16.2 Timestamp Elements

2291 The specification defines XML elements which may be used to express timestamp information
2292 such as creation and expiration. While defined in the context of message security, these
2293 elements can be re-used wherever these sorts of time statements need to be made.

2294

2295 The elements in this specification are defined and illustrated using time references in terms of the
2296 *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this
2297 type for interoperability. It is further RECOMMENDED that all references be in UTC time for
2298 increased interoperability. If, however, other time types are used, then the `valueType` attribute
2299 MUST be specified to indicate the data type of the time format.

2300

The following table provides an overview of these elements:

2301

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.

2302

2303

A detailed description can be found in Section 10.

2304
2305
2306

This section is non-normative.

2307

16.3 General Schema Types

2308
2309
2310
2311
2312
2313
2314

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

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 <code>wsu:Id</code> attribute as well as extensibility for other namespace qualified attributes.
wsu:AttributedDateTime type	This type extends the XML Schema <code>dateTime</code> type to include the common attributes.
wsu:AttributedURI type	This type extends the XML Schema <code>anyURI</code> type to include the common attributes.

2315
2316
2317

This section is non-normative.

2318

Appendix D: SecurityTokenReference Model

2319

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

2320

2321

2322

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

2323

2324

- The XML Signature reference mechanisms are focused on "key" references rather than general token references.
- The XML Signature reference mechanisms utilize a fairly closed schema which limits the extensibility that can be applied.
- There are additional types of general reference mechanisms that are needed, but are not covered by XML Signature.
- There are scenarios where a reference may occur outside of an XML Signature and the XML Signature schema is not appropriate or desired.
- The XML Signature references may include aspects (e.g. transforms) that may not apply to all references.

2325

2326

2327

2328

2329

2330

2331

2332

2333

2334

The following use cases drive the above motivations:

2335

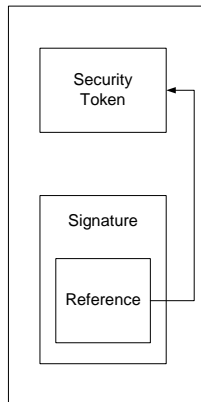
2336

2337

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

2338

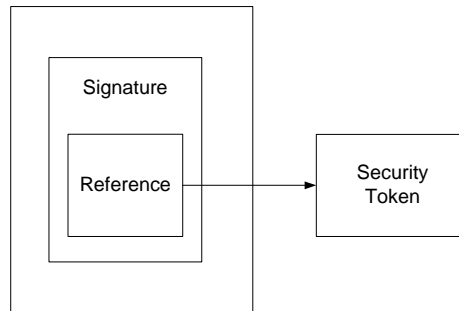
header, is associated with an XML Signature. The figure below illustrates this:



2339

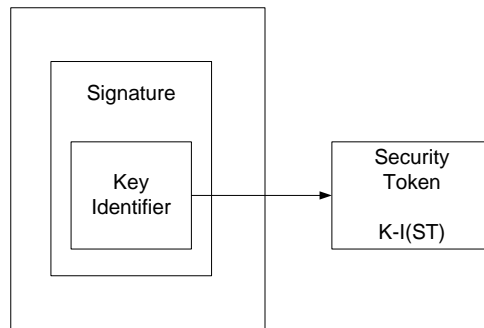
2340
2341
2342
2343

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:



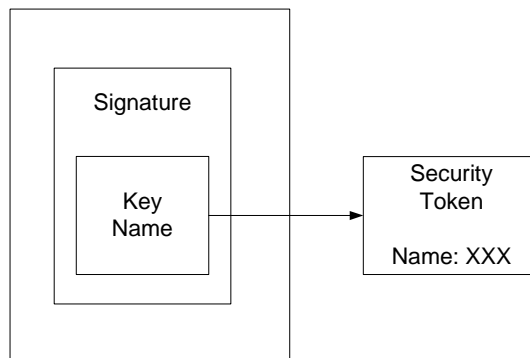
2344
2345
2346
2347

Key Identifier – A security token, which is associated with an XML Signature and identified using a known value that is the result of a well-known function of the security token (defined by the token format or profile). The figure below illustrates this where the token is located externally:



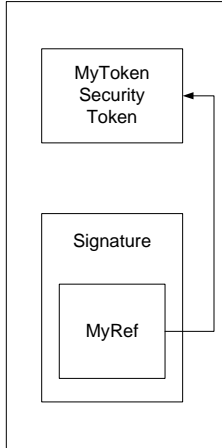
2348
2349
2350
2351

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

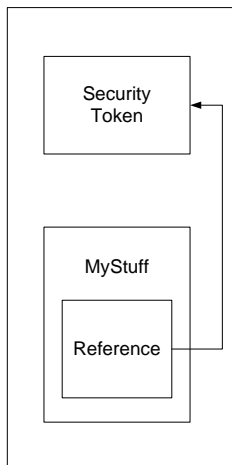


2352
2353
2354
2355
2356

Format-Specific References – A security token is associated with an XML Signature and identified using a mechanism specific to the token (rather than the general mechanisms described above). The figure below illustrates this:



2357 **Non-Signature References** – A message may contain XML that does not represent an XML
 2358 signature, but may reference a security token (which may or may not be included in the
 2359 message). The figure below illustrates this:



2360

2361

2362 All conformant implementations **MUST** be able to process the
 2363 `<wsse:SecurityTokenReference>` element. However, they are not required to support all of
 2364 the different types of references.

2365

2366 The reference **MAY** include a *ValueType* attribute which provides a "hint" for the type of desired
 2367 token.

2368

2369 If multiple sub-elements are specified, together they describe the reference for the token.

2370 There are several challenges that implementations face when trying to interoperate:

2371

2372 **ID References** – The underlying XML referencing mechanism using the XML base type of ID
 2373 provides a simple straightforward XML element reference. However, because this is an XML
 2374 type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
 2375 requires the recipient to *understand* the schema. This may be an expensive task and in the
 2376 general case impossible as there is no way to know the "schema location" for a specific
 2377 namespace URI.

2377

2378 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
 2379 references are, by definition, unique by XML. However, other mechanisms such as "principal
 2380 name" are not required to be unique and therefore such references may be unique.

2381

2382 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
 2383 information about the "key" used in the signature. For token references within signatures, it is
 2384 RECOMMENDED that the `<wsse:SecurityTokenReference>` be placed within the

2384

2385 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
2386 Message Security or its profiles are preferred over the mechanisms in XML Signature.
2387 The following provides additional details on the specific reference mechanisms defined in WSS:
2388 SOAP Message Security:
2389

2390 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
2391 the security token. If only the fragment is specified, then it references the security token within
2392 the document whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to
2393 a [potentially external] security token identified using a URI. There are no implied semantics
2394 around the processing of the URI.
2395

2396 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
2397 by specifying a known value (identifier) for the token, which is determined by applying a special
2398 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
2399 specific security token but requires a profile or token-specific function to be specified. The
2400 `ValueType` attribute defines the type of key identifier and, consequently, identifies the type of
2401 token referenced. The `EncodingType` attribute specifies how the unique value (identifier) is
2402 encoded. For example, a hash value may be encoded using base 64 encoding.
2403

2404 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
2405 specific value that is used to *match* an identity assertion within the security token. This is a
2406 subset match and may result in multiple security tokens that match the specified name. While
2407 XML Signature doesn't imply formatting semantics, WSS: SOAP Message Security
2408 RECOMMENDS that X.509 names be specified.
2409

2410 It is expected that, where appropriate, profiles define if and how the reference mechanisms map
2411 to the specific token profile. Specifically, the profile should answer the following questions:
2412

- 2413 • What types of references can be used?
 - 2414 • How "Key Name" references map (if at all)?
 - 2415 • How "Key Identifier" references map (if at all)?
 - 2416 • Are there any additional profile or format-specific references?
- 2417

2418 This section is non-normative.