



1
2

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

6 **Working Draft - 07 November 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/oasis-wss-soap-message-security-1.1>

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
34
35
36

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

37
38
39
40
41
42
43
44
45
46
47
48

Status:

This is a technical committee document submitted for consideration by the OASIS Web Services Security (WSS) technical committee. Please send comments to the editors. If you are on the wss@lists.oasis-open.org list for committee members, send comments there. If you are not on that list, subscribe to the wss-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to wss-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message. For patent disclosure information that may be essential to the implementation of this specification, and any offers of licensing terms, refer to the Intellectual Property Rights section of the OASIS Web Services Security Technical Committee (WSS TC) web page at <http://www.oasis-open.org/committees/wss/ipr.php>. General OASIS IPR information 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 implementors or users of this specification, can be
59 obtained from the OASIS Executive Director. OASIS invites any interested party to bring to its
60 attention any copyrights, patents or patent applications, or other proprietary rights which may
61 cover technology that may be required to implement this specification. Please address the
62 information to the OASIS Executive Director.

63

64 Copyright (C) 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 may not be modified in any way, such as by removing the copyright
71 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 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
86 contents of this specification. For more information consult the online list of claimed rights.

87

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

Table of Contents

90	1	Introduction	7
91	1.1	Goals and Requirements	7
92	1.1.1	Requirements.....	8
93	1.1.2	Non-Goals	8
94	2	Notations and Terminology.....	9
95	2.1	Notational Conventions	9
96	2.2	Namespaces	9
97	2.3	Acronyms and Abbreviations	10
98	2.4	Terminology.....	11
99	2.5	Note on Examples	12
100	3	Message Protection Mechanisms.....	13
101	3.1	Message Security Model.....	13
102	3.2	Message Protection.....	13
103	3.3	Invalid or Missing Claims	14
104	3.4	Example	14
105	4	ID References	17
106	4.1	Id Attribute	17
107	4.2	Id Schema	18
108	5	Security Header	20
109	6	Security Tokens	23
110	6.1	Attaching Security Tokens	23
111	6.1.1	Processing Rules	23
112	6.1.2	Subject Confirmation.....	23
113	6.2	User Name Token	23
114	6.2.1	Usernames.....	23
115	6.3	Binary Security Tokens	24
116	6.3.1	Attaching Security Tokens	24
117	6.3.2	Encoding Binary Security Tokens.....	24
118	6.4	XML Tokens	26
119	6.5	EncryptedData Token	26
120	6.6	Identifying and Referencing Security Tokens	26
121	7	Token References.....	27
122	7.1	SecurityTokenReference Element	27
123	7.2	Direct References.....	29

124	7.3 Key Identifiers.....	30
125	7.4 Embedded References	32
126	7.5 ds:KeyInfo	33
127	7.6 Key Names.....	33
128	7.7 Encrypted Key reference.....	34
129	8 Signatures.....	35
130	8.1 Algorithms	35
131	8.2 Signing Messages.....	38
132	8.3 Signing Tokens.....	38
133	8.4 Signature Validation	41
134	8.5 Signature Confirmation	42
135	8.5.1 Response Generation Rules.....	43
136	8.5.2 Response Processing Rules.....	43
137	8.6 Example	44
138	9 Encryption	45
139	9.1 xenc:ReferenceList	45
140	9.2 xenc:EncryptedKey	46
141	9.3 Encrypted Header	47
142	9.4 Processing Rules	47
143	9.4.1 Encryption	48
144	9.4.2 Decryption.....	48
145	9.4.3 Encryption with EncryptedHeader	49
146	9.4.4 Processing an EncryptedHeader	49
147	9.4.5 Processing the mustUnderstand attribute on EncryptedHeader	50
148	10 Security Timestamps	51
149	11 Extended Example.....	54
150	12 Error Handling.....	57
151	13 Security Considerations	59
152	13.1 General Considerations	59
153	13.2 Additional Considerations	59
154	13.2.1 Replay.....	59
155	13.2.2 Combining Security Mechanisms	60
156	13.2.3 Challenges.....	60
157	13.2.4 Protecting Security Tokens and Keys.....	60
158	13.2.5 Protecting Timestamps and Ids	61
159	13.2.6 Protecting against removal and modification of XML Elements	61
160	13.2.7 Detecting Duplicate Identifiers	62
161	14 Interoperability Notes	63

162	15	Privacy Considerations	64
163	16	References.....	65
164		Appendix A: Acknowledgements.....	67
165		Appendix B: Revision History	70
166		Appendix C: Utility Elements and Attributes.....	71
167	16.1	Identification Attribute.....	71
168	16.2	Timestamp Elements	71
169	16.3	General Schema Types	72
170		Appendix D: SecurityTokenReference Model	73
171			

172

1 Introduction

173 This OASIS specification is the result of significant new work by the WSS Technical Committee
174 and supersedes the input submissions, Web Service Security (WS-Security) Version 1.0 April 5,
175 2002 and Web Services Security Addendum Version 1.0 August 18, 2002.

176

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

181

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

187

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

194

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

1.1 Goals and Requirements

198
199 The goal of this specification is to enable applications to conduct secure SOAP message
200 exchanges.

201

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

205

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

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

213

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

215 **1.1.1 Requirements**

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

- 218 • Multiple security token formats
- 219 • Multiple trust domains
- 220 • Multiple signature formats
- 221 • Multiple encryption technologies
- 222 • End-to-end message content security and not just transport-level security

223 **1.1.2 Non-Goals**

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

225

- 226 • Establishing a security context or authentication mechanisms.
- 227 • Key derivation.
- 228 • Advertisement and exchange of security policy.
- 229 • How trust is established or determined.
- 230 • Non-repudiation.

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

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/oasis-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.

272 The namespaces used in this document are shown in the following table (note that for brevity, the
 273 examples use the prefixes listed below but do not include the URIs – those listed below are
 274 assumed).
 275

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
wssell	http://docs.oasis-open.org/wss/oasis-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#

276
 277 The URLs provided for the `wsse` and `wsu` namespaces can be used to obtain the schema files.
 278
 279 URI fragments defined in this document are relative to the following base URI unless otherwise
 280 stated:
 281 <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0>

282 2.3 Acronyms and Abbreviations

283 The following (non-normative) table defines acronyms and abbreviations for this document.
 284

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

285
286
287
288
289
290
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
325
326

2.4 Terminology

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

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

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

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 – A *digital 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 digital 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 digital 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).

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.

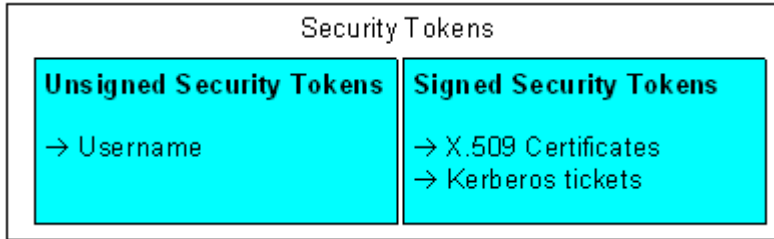
Integrity – *Integrity* is the property that data has not been modified.

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 - In this document, signature and digital signature are used interchangeably and have the same meaning.

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



327
328

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

331

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

334 2.5 Note on Examples

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

338

339 Specifically, the examples are constrained to contain only mechanisms defined in this document.
340 The only reason for this is to avoid requiring the reader to consult other documents merely to
341 understand the examples. It is NOT intended to suggest that the mechanisms illustrated
342 represent best practice or are the strongest available to implement the security properties in
343 question. In particular, mechanisms defined in other Token Profiles are known to be stronger,
344 more efficient and/or generally superior to some of the mechanisms shown in the examples in this
345 document.

346

347

3 Message Protection Mechanisms

348

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

349

350

351

- the message could be modified or read by attacker 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.

352

353

354

355

356

357

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

358

3.1 Message Security Model

359

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

360

361

362

363

364

365

366

367

368

369

370

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.

371

372

373

374

375

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.

376

377

378

379

380

381

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.

382

3.2 Message Protection

383

384

385

386

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).

387
388 Message integrity is provided by XML Signature [XMLSIG] in conjunction with security tokens to
389 ensure that modifications to messages are detected. The integrity mechanisms are designed to
390 support multiple signatures, potentially by multiple SOAP actors/roles, and to be extensible to
391 support additional signature formats.
392
393 Message confidentiality leverages XML Encryption [XMLENC] in conjunction with security tokens
394 to keep portions of a SOAP message confidential. The encryption mechanisms are designed to
395 support additional encryption processes and operations by multiple SOAP actors/roles.
396
397 This document defines syntax and semantics of signatures within a <wsse:Security> element.
398 This document does not constrain any signature appearing outside of a <wsse:Security>
399 element.

400 3.3 Invalid or Missing Claims

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

408 3.4 Example

409 The following example illustrates the use of a custom security token and associated signature.
410 The token contains base64 encoded binary data conveying a symmetric key which, we assume,
411 can be properly authenticated by the recipient. The message producer uses the symmetric key
412 with an HMAC signing algorithm to sign the message. The message receiver uses its knowledge
413 of the shared secret to repeat the HMAC key calculation which it uses to validate the signature
414 and in the process confirm that the message was authored by the claimed user identity.
415

```
416 (001) <?xml version="1.0" encoding="utf-8"?>  
417 (002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."  
418         xmlns:ds="...">  
419 (003)   <S11:Header>  
420 (004)     <wsse:Security  
421         xmlns:wsse="...">  
422 (005)       <wsse:BinarySecurityToken ValueType="  
423 http://fabrikam123#CustomToken "  
424         EncodingType="...#Base64Binary" wsu:Id=" MyID ">  
425 (006)         FHUIORv...  
426 (007)       </wsse:BinarySecurityToken>  
427 (008)       <ds:Signature>  
428 (009)         <ds:SignedInfo>  
429 (010)           <ds:CanonicalizationMethod  
430             Algorithm=  
431               "http://www.w3.org/2001/10/xml-exc-c14n#" />  
432 (011)           <ds:SignatureMethod
```

```

433         Algorithm=
434         "http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
435 (012)     <ds:Reference URI="#MsgBody">
436 (013)     <ds:DigestMethod
437         Algorithm=
438         "http://www.w3.org/2000/09/xmldsig#sha1" />
439 (014)     <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
440 (015)     </ds:Reference>
441 (016)     </ds:SignedInfo>
442 (017)     <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
443 (018)     <ds:KeyInfo>
444 (019)         <wsse:SecurityTokenReference>
445 (020)             <wsse:Reference URI="#MyID" />
446 (021)         </wsse:SecurityTokenReference>
447 (022)     </ds:KeyInfo>
448 (023)     </ds:Signature>
449 (024)     </wsse:Security>
450 (025) </S11:Header>
451 (026) <S11:Body wsu:Id="MsgBody">
452 (027)     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
453         QQQ
454     </tru:StockSymbol>
455 (028) </S11:Body>
456 (029) </S11:Envelope>

```

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

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

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

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

470
471 Lines (009) to (016) describe what is being signed and the type of canonicalization being used.

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

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

481

482 Lines (018) to (022) provides information, partial or complete, as to where to find the security
483 token associated with this signature. Specifically, lines (019) to (021) indicate that the security
484 token can be found at (pulled from) the specified URL.
485
486 Lines (026) to (028) contain the body (payload) of the SOAP message.
487

488

4 ID References

489 There are many motivations for referencing other message elements such as signature
490 references or correlating signatures to security tokens. For this reason, this specification defines
491 the `wsu:Id` attribute so that recipients need not understand the full schema of the message for
492 processing of the security elements. That is, they need only "know" that the `wsu:Id` attribute
493 represents a schema type of ID which is used to reference elements. However, because some
494 key schemas used by this specification don't allow attribute extensibility (namely XML Signature
495 and XML Encryption), this specification also allows use of their local ID attributes in addition to
496 the `wsu:Id` attribute and the `xml:id` attribute [XMLID]. As a consequence, when trying to locate
497 an element referenced in a signature, the following attributes are considered (in no particular
498 order):

499

- 500 • Local ID attributes on XML Signature elements
- 501 • Local ID attributes on XML Encryption elements
- 502 • Global `wsu:Id` attributes (described below) on elements
- 503 • Profile specific defined identifiers
- 504 • Global `xml:id` attributes on elements

505

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

509

510 Tokens and elements that are defined in this specification and related profiles to use `wsu:Id`
511 attributes SHOULD use `wsu:Id`. Elements to be signed MAY use `xml:id` [XMLID] or `wsu:Id`,
512 and use of `xml:id` MAY be specified in profiles. All receivers MUST be able to identify XML
513 elements carrying a `wsu:Id` attribute as representing an attribute of schema type ID and process
514 it accordingly.

515

516 All receivers MAY be able to identify XML elements with a `xml:id` attribute as representing an ID
517 attribute and process it accordingly. Senders SHOULD use `wsu:Id` and MAY use `xml:id`. Note
518 that use of `xml:id` in conjunction with inclusive canonicalization may be inappropriate, as noted
519 in [XMLID] and thus this combination SHOULD be avoided.

520

4.1 Id Attribute

521
522 There are many situations where elements within SOAP messages need to be referenced. For
523 example, when signing a SOAP message, selected elements are included in the scope of the
524 signature. XML Schema Part 2 [XMLSCHEMA] provides several built-in data types that may be
525 used for identifying and referencing elements, but their use requires that consumers of the SOAP
526 message either have or must be able to obtain the schemas where the identity or reference
527 mechanisms are defined. In some circumstances, for example, intermediaries, this can be
528 problematic and not desirable.

529

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

534

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

538

539 Alternatively, the `xml:id` attribute MAY be used. Applications MUST NOT specify both a
540 `wsu:Id` and `xml:id` attribute on a single element. It is an XML requirement that only one id
541 attribute be specified on a single element.

542

4.2 Id Schema

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

546 The syntax for this attribute is as follows:

547

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

549

550 The following describes the attribute illustrated above:

551 `.../@wsu:Id`

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

554

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

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

559

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

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

563

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

566

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

569

570 Conformant processors that do not support dynamic XML Schema or DTDs discovery and
571 processing are strongly encouraged to integrate this attribute definition into their parsers. That is,
572 to treat this attribute information item as if its PSVI has a [type definition] which {target
573 namespace} is "`http://www.w3.org/2001/XMLSchema`" and which {type} is "ID." Doing so
574 allows the processor to inherently know *how* to process the attribute without having to locate and

575 process the associated schema. Specifically, implementations MAY support the value of the
576 `wsu:Id` as the valid identifier for use as an XPointer [XPointer] shorthand pointer for
577 interoperability with XML Signature references.

578

5 Security Header

579 The `<wsse:Security>` header block provides a mechanism for attaching security-related
580 information targeted at a specific recipient in the form of a SOAP actor/role. This may be either
581 the ultimate recipient of the message or an intermediary. Consequently, elements of this type
582 may be present multiple times in a SOAP message. An active intermediary on the message path
583 MAY add one or more new sub-elements to an existing `<wsse:Security>` header block if they
584 are targeted for its SOAP node or it MAY add one or more new headers for additional targets.
585

586 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted
587 for separate recipients. A message MUST NOT have multiple `<wsse:Security>` header blocks
588 targeted (whether explicitly or implicitly) at the same recipient. However, only one
589 `<wsse:Security>` header block MAY omit the `S11:actor` or `S12:role` attributes. Two
590 `<wsse:Security>` header blocks MUST NOT have the same value for `S11:actor` or
591 `S12:role`. Message security information targeted for different recipients MUST appear in
592 different `<wsse:Security>` header blocks. This is due to potential processing order issues
593 (e.g. due to possible header re-ordering). The `<wsse:Security>` header block without a
594 specified `S11:actor` or `S12:role` MAY be processed by anyone, but MUST NOT be removed
595 prior to the final destination or endpoint.
596

597 As elements are added to a `<wsse:Security>` header block, they SHOULD be prepended to
598 the existing elements. As such, the `<wsse:Security>` header block represents the signing and
599 encryption steps the message producer took to create the message. This prepending rule
600 ensures that the receiving application can process sub-elements in the order they appear in the
601 `<wsse:Security>` header block, because there will be no forward dependency among the sub-
602 elements. Note that this specification does not impose any specific order of processing the sub-
603 elements. The receiving application can use whatever order is required.
604

605 When a sub-element refers to a key carried in another sub-element (for example, a signature
606 sub-element that refers to a binary security token sub-element that contains the X.509 certificate
607 used for the signature), the key-bearing element SHOULD be ordered to precede the key-using
608 Element:
609

610
611
612
613
614
615
616
617
618
619
620
621

```
<S11:Envelope>
  <S11:Header>
    ...
    <wsse:Security S11:actor="..." S11:mustUnderstand="...">
      ...
    </wsse:Security>
    ...
  </S11:Header>
  ...
</S11:Envelope>
```

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

622 */wsse:Security*
623 This is the header block for passing security-related message information to a recipient.
624
625 */wsse:Security/@S11:actor*
626 This attribute allows a specific SOAP 1.1 [SOAP11] actor to be identified. This attribute
627 is optional; however, no two instances of the header block may omit an actor or specify
628 the same actor.
629
630 */wsse:Security/@S12:role*
631 This attribute allows a specific SOAP 1.2 [SOAP12] role to be identified. This attribute is
632 optional; however, no two instances of the header block may omit a role or specify the
633 same role.
634
635 */wsse:Security/@S11:mustUnderstand*
636 This SOAP 1.1 [SOAP11] attribute is used to indicate whether a header entry is
637 mandatory or optional for the recipient to process. The value of the mustUnderstand
638 attribute is either "1" or "0". The absence of the SOAP mustUnderstand attribute is
639 semantically equivalent to its presence with the value "0".
640
641 */wsse:Security/@S12:mustUnderstand*
642 This SOAP 1.2 [SPOAP12] attribute is used to indicate whether a header entry is
643 mandatory or optional for the recipient to process. The value of the mustUnderstand
644 attribute is either "true", "1" "false" or "0". The absence of the SOAP mustUnderstand
645 attribute is semantically equivalent to its presence with the value "false".
646
647 */wsse:Security/{any}*
648 This is an extensibility mechanism to allow different (extensible) types of security
649 information, based on a schema, to be passed. Unrecognized elements SHOULD cause
650 a fault.
651
652 */wsse:Security/@{any}*
653 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
654 added to the header. Unrecognized attributes SHOULD cause a fault.
655
656 All compliant implementations MUST be able to process a `<wsse:Security>` element.
657
658 All compliant implementations MUST declare which profiles they support and MUST be able to
659 process a `<wsse:Security>` element including any sub-elements which may be defined by that
660 profile. It is RECOMMENDED that undefined elements within the `<wsse:Security>` header
661 not be processed.
662
663 The next few sections outline elements that are expected to be used within a `<wsse:Security>`
664 header.
665
666 When a `<wsse:Security>` header includes a `mustUnderstand="true"` attribute:
667

- The receiver MUST generate a SOAP fault if does not implement the WSS: SOAP
668 Message Security specification corresponding to the namespace. Implementation means

669 ability to interpret the schema as well as follow the required processing rules specified in
670 WSS: SOAP Message Security.
671 • The receiver MUST generate a fault if unable to interpret or process security tokens
672 contained in the <wsse:Security> header block according to the corresponding WSS:
673 SOAP Message Security token profiles.
674 • Receivers MAY ignore elements or extensions within the <wsse:Security> element,
675 based on local security policy.

676

6 Security Tokens

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

6.1 Attaching Security Tokens

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

683

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

6.1.1 Processing Rules

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

6.1.2 Subject Confirmation

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

6.2 User Name Token

6.2.1 Usernames

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

699 The following illustrates the syntax of this element:

700

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

704

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

706

707 */wsse:UsernameToken*

708 This element is used to represent a claimed identity.

709

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

711 A string label for this security token. The `wsu:Id` allow for an open attribute model.
712
713 */wsse:UsernameToken/wsse:Username*
714 This required element specifies the claimed identity.
715
716 */wsse:UsernameToken/wsse:Username/@{any}*
717 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
718 added to the `<wsse:Username>` element.
719
720 */wsse:UsernameToken/{any}*
721 This is an extensibility mechanism to allow different (extensible) types of security
722 information, based on a schema, to be passed. Unrecognized elements SHOULD cause
723 a fault.
724
725 */wsse:UsernameToken/@{any}*
726 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
727 added to the `<wsse:UsernameToken>` element. Unrecognized attributes SHOULD
728 cause a fault.
729
730 All compliant implementations MUST be able to process a `<wsse:UsernameToken>`
731 element.
732 The following illustrates the use of this:

```
733 <S11:Envelope xmlns:S11="..." xmlns:wsse="...">  
734   <S11:Header>  
735     ...  
736     <wsse:Security>  
737       <wsse:UsernameToken>  
738         <wsse:Username>Zoe</wsse:Username>  
739       </wsse:UsernameToken>  
740     </wsse:Security>  
741     ...  
742   </S11:Header>  
743   ...  
744 </S11:Envelope>
```

747 6.3 Binary Security Tokens

748 6.3.1 Attaching Security Tokens

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

752 6.3.2 Encoding Binary Security Tokens

753 Binary security tokens (e.g., X.509 certificates and Kerberos [KERBEROS] tickets) or other non-
754 XML formats require a special encoding format for inclusion. This section describes a basic

755 framework for using binary security tokens. Subsequent specifications MUST describe the rules
756 for creating and processing specific binary security token formats.

757

758 The <wsse:BinarySecurityToken> element defines two attributes that are used to interpret
759 it. The `ValueType` attribute indicates what the security token is, for example, a Kerberos ticket.
760 The `EncodingType` tells how the security token is encoded, for example `Base64Binary`.

761 The following is an overview of the syntax:

762

```
<wsse:BinarySecurityToken wsu:Id=...  
                          EncodingType=...  
                          ValueType=.../>
```

766

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

768 */wsse:BinarySecurityToken*

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

770

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

772 An optional string label for this security token.

773

774 */wsse:BinarySecurityToken/@ValueType*

775 The `ValueType` attribute is used to indicate the "value space" of the encoded binary
776 data (e.g. an X.509 certificate). The `ValueType` attribute allows a URI that defines the
777 value type and space of the encoded binary data. Subsequent specifications MUST
778 define the `ValueType` value for the tokens that they define. The usage of `ValueType` is
779 RECOMMENDED.

780

781 */wsse:BinarySecurityToken/@EncodingType*

782 The `EncodingType` attribute is used to indicate, using a URI, the encoding format of the
783 binary data (e.g., `base64` encoded). A new attribute is introduced, as there are issues
784 with the current schema validation tools that make derivations of mixed simple and
785 complex types difficult within XML Schema. The `EncodingType` attribute is interpreted
786 to indicate the encoding format of the element. The following encoding formats are pre-
787 defined:

788

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

789

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

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

793

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

796 **6.4 XML Tokens**

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

799 **6.5 EncryptedData Token**

800 In certain cases it is desirable that the token included in the `<wsse:Security>` header be
801 encrypted for the recipient processing role. In such a case the `<xenc:EncryptedData>`
802 element MAY be used to contain a security token and included in the `<wsse:Security>`
803 header. That is this specification defines the usage of `<xenc:EncryptedData>` to encrypt
804 security tokens contained in `<wsse:Security>` header.
805

806 It should be noted that token references are not made to the `<xenc:EncryptedData>` element,
807 but instead to the token represented by the clear-text, once the `<xenc:EncryptedData>`
808 element has been processed (decrypted). Such references utilize the token profile for the
809 contained token. i.e., `<xenc:EncryptedData>` SHOULD NOT include an XML ID for
810 referencing the contained security token.
811

812 All `<xenc:EncryptedData>` tokens SHOULD either have an embedded encryption key or
813 should be referenced by a separate encryption key.

814 When a `<xenc:EncryptedData>` token is processed, it is replaced in the message infoset with
815 its decrypted form.

816 **6.6 Identifying and Referencing Security Tokens**

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

822

7 Token References

823 This chapter discusses and defines mechanisms for referencing security tokens and other key
824 bearing elements..

825 7.1 SecurityTokenReference Element

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

830

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

835

836 If a `<wsse:SecurityTokenReference>` is used outside of the security header processing
837 block the meaning of the response and/or processing rules of the resulting references MUST be
838 specified by the the specific profile and are out of scope of this specification.

839 The following illustrates the syntax of this element:

840

```
841 <wsse:SecurityTokenReference wsu:Id="...", wss11:TokenType="...",  
842 wsse:Usage="...", wsse:Usage="...">  
843 </wsse:SecurityTokenReference>
```

844

845 The following describes the elements defined above:

846

847 */wsse:SecurityTokenReference*

848 This element provides a reference to a security token.

849

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

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

855

856 */wsse:SecurityTokenReference/@wsse11:TokenType*

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

861

862 When a `wss11:TokenType` attribute is specified in conjunction with a
 863 `wsse:KeyIdentifier/@ValueType` attribute or a `wsse:Reference/@ValueType`
 864 attribute that indicates the type of the referenced token, the security token type identified
 865 by the `wss11:TokenType` attribute MUST be consistent with the security token type
 866 identified by the `wsse:ValueType` attribute.
 867

URI	Description
http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey	A token type of an <code><xenc:EncryptedKey></code>

868
 869 `/wsse:SecurityTokenReference/@wsse:Usage`
 870 This optional attribute is used to type the usage of the
 871 `<wsse:SecurityTokenReference>`. Usages are specified using URIs and multiple
 872 usages MAY be specified using XML list semantics. No usages are defined by this
 873 specification.
 874
 875 `/wsse:SecurityTokenReference/{any}`
 876 This is an extensibility mechanism to allow different (extensible) types of security
 877 references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
 878 fault.
 879
 880 `/wsse:SecurityTokenReference/@{any}`
 881 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 882 added to the header. Unrecognized attributes SHOULD cause a fault.
 883
 884 All compliant implementations MUST be able to process a
 885 `<wsse:SecurityTokenReference>` element.
 886
 887 This element can also be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to
 888 retrieve the key information from a security token placed somewhere else. In particular, it is
 889 RECOMMENDED, when using XML Signature and XML Encryption, that a
 890 `<wsse:SecurityTokenReference>` element be placed inside a `<ds:KeyInfo>` to reference
 891 the security token used for the signature or encryption.
 892
 893 There are several challenges that implementations face when trying to interoperate. Processing
 894 the IDs and references requires the recipient to *understand* the schema. This may be an
 895 expensive task and in the general case impossible as there is no way to know the "schema
 896 location" for a specific namespace URI. As well, the primary goal of a reference is to uniquely
 897 identify the desired token. ID references are, by definition, unique by XML. However, other
 898 mechanisms such as "principal name" are not required to be unique and therefore such
 899 references may be not unique.
 900

901 This specification allows for the use of multiple reference mechanisms within a single
902 <wsse:SecurityTokenReference>. When multiple references are present in a given
903 <wsse:SecurityTokenReference>, they MUST resolve to a single token in common.
904 Specific token profiles SHOULD define the reference mechanisms to be used.

905
906 The following list provides a list of the specific reference mechanisms defined in WSS: SOAP
907 Message Security in preferred order (i.e., most specific to least specific):
908

- 909 • **Direct References** – This allows references to included tokens using URI fragments and
910 external tokens using full URIs.
- 911 • **Key Identifiers** – This allows tokens to be referenced using an opaque value that
912 represents the token (defined by token type/profile).
- 913 • **Key Names** – This allows tokens to be referenced using a string that matches an identity
914 assertion within the security token. This is a subset match and may result in multiple
915 security tokens that match the specified name.
- 916 • **Embedded References** - This allows tokens to be embedded (as opposed to a pointer
917 to a token that resides elsewhere).

918 7.2 Direct References

919 The <wsse:Reference> element provides an extensible mechanism for directly referencing
920 security tokens using URIs.

921
922 The following illustrates the syntax of this element:

```
923 <wsse:SecurityTokenReference wsu:Id="...">  
924   <wsse:Reference URI="..." ValueType="..." />  
925 </wsse:SecurityTokenReference>
```

926
927
928 The following describes the elements defined above:

929
930 */wsse:SecurityTokenReference/wsse:Reference*

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

932

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

934 This optional attribute specifies an abstract URI for a security token. If a fragment is
935 specified, then it indicates the local ID of the security token being referenced. The URI
936 MUST identify a security token. The URI MUST NOT identify a
937 *wsse:SecurityTokenReference* element, a *wsse:Embedded* element, a
938 *wsse:Reference* element, or a *wsse:KeyIdentifier* element.
939

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

941 This optional attribute specifies a URI that is used to identify the *type* of token being
942 referenced. This specification does not define any processing rules around the usage of
943 this attribute, however, specifications for individual token types MAY define specific
944 processing rules and semantics around the value of the URI and its interpretation. If this
945 attribute is not present, the URI MUST be processed as a normal URI.
946

947 In this version of the specification the use of this attribute to identify the type of the
948 referenced security token is deprecated. Profiles which require or recommend the use of
949 this attribute to identify the type of the referenced security token SHOULD evolve to
950 require or recommend the use of the
951 `wsse:SecurityTokenReference/@wsse11:TokenType` attribute to identify the type
952 of the referenced token.

953
954 `/wsse:SecurityTokenReference/wsse:Reference/{any}`

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

958
959 `/wsse:SecurityTokenReference/wsse:Reference/@{any}`

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

962
963 The following illustrates the use of this element:

```
964  
965 <wsse:SecurityTokenReference  
966     xmlns:wsse="...">  
967   <wsse:Reference  
968     URI="http://www.fabrikam123.com/tokens/Zoe"/>  
969 </wsse:SecurityTokenReference>
```

970 7.3 Key Identifiers

971 Alternatively, if a direct reference is not used, then it is RECOMMENDED that a key identifier be
972 used to specify/reference a security token instead of a `<ds:KeyName>`. A
973 `<wsse:KeyIdentifier>` is a value that can be used to uniquely identify a security token (e.g. a
974 hash of the important elements of the security token). The exact value type and generation
975 algorithm varies by security token type (and sometimes by the data within the token),
976 Consequently, the values and algorithms are described in the token-specific profiles rather than
977 this specification.

978
979 The `<wsse:KeyIdentifier>` element SHALL be placed in the
980 `<wsse:SecurityTokenReference>` element to reference a token using an identifier. This
981 element SHOULD be used for all key identifiers.

982
983 The processing model assumes that the key identifier for a security token is constant.
984 Consequently, processing a key identifier involves simply looking for a security token whose key
985 identifier matches the specified constant. The `<wsse:KeyIdentifier>` element is only allowed
986 inside a `<wsse:SecurityTokenReference>` element

987 The following is an overview of the syntax:

```
988  
989 <wsse:SecurityTokenReference>  
990   <wsse:KeyIdentifier wsu:Id="..."  
991     ValueType="..."  
992     EncodingType="...">
```

993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013

```
...  
</wsse:KeyIdentifier>  
</wsse:SecurityTokenReference>
```

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

/wsse:SecurityTokenReference/wsse:KeyIdentifier

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

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

An optional string label for this identifier.

/wsse:SecurityTokenReference/wsse:KeyIdentifier/@ValueType

The optional `ValueType` attribute is used to indicate the type of `KeyIdentifier` being used. This specification defines one `ValueType` that can be applied to all token types. Each specific token profile specifies the `KeyIdentifier` types that may be used to refer to tokens of that type. It also specifies the critical semantics of the identifier, such as whether the `KeyIdentifier` is unique to the key or the token. If no value is specified then the key identifier will be interpreted in an application-specific manner. This URI fragment is relative to a base URI as indicated in the table below.

URI	Description
<code>http://docs.oasis-open.org/wss/oasis-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. A thumbprint reference MUST occur in combination with a required to be supported (by the applicable profile) reference form unless a thumbprint reference is among the reference forms required to be supported by the applicable profile, or the parties to the communication have agreed to accept thumbprint only references.
<code>http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKeySHA1</code>	If the security token type that the Security Token Reference refers to already contains a representation for the <code>EncryptedKey</code> , the value obtained from the token MAY be used. If the token does not contain a representation of a <code>EncryptedKey</code> , 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.

1014
1015
1016
1017
1018
1019
1020
1021
1022

/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:

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

1023
1024
1025
1026

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

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

1027

7.4 Embedded References

1028
1029
1030
1031
1032

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

The following is an overview of the syntax:

1033
1034
1035
1036
1037
1038

```
<wsse:SecurityTokenReference>  
  <wsse:Embedded wsu:Id="...">  
    ...  
  </wsse:Embedded>  
</wsse:SecurityTokenReference>
```

1039

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

1040
1041

/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).

1044
1045

/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.

1047
1048

/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.

1049
1050
1051
1052

1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075

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

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

The following example illustrates embedding a SAML assertion:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
  <S11:Header>
    <wsse:Security>
      ...
      <wsse:SecurityTokenReference>
        <wsse:Embedded wsu:Id="tok1">
          <saml:Assertion xmlns:saml="...">
            ...
          </saml:Assertion>
        </wsse:Embedded>
      </wsse:SecurityTokenReference>
      ...
    </wsse:Security>
  </S11:Header>
  ...
</S11:Envelope>
```

1076

7.5 ds:KeyInfo

1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087

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

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

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

1088

7.6 Key Names

1089
1090
1091
1092
1093
1094
1095

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

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

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

1096 **7.7 Encrypted Key reference**

1097 In certain cases, an `<xenc:EncryptedKey>` element MAY be used to carry key material
1098 encrypted for the recipient's key. This key material is henceforth referred to as `EncryptedKey`.

1099
1100 The `EncryptedKey` MAY be used to perform other cryptographic operations within the same
1101 message, such as signatures. The `EncryptedKey` MAY also be used for performing
1102 cryptographic operations in subsequent messages exchanged by the two parties. Two
1103 mechanisms are defined for referencing the `EncryptedKey`.

1104
1105 When referencing the `EncryptedKey` within the same message that contains the
1106 `<xenc:EncryptedKey>` element, the `<ds:KeyInfo>` element of the referencing construct
1107 MUST contain a `<wsse:SecurityTokenReference>`. The
1108 `<wsse:SecurityTokenReference>` element MUST contain a `<wsse:Reference>` element.

1109
1110 The URI attribute value of the `<wsse:Reference>` element MUST be set to the value of the ID
1111 attribute of the referenced `<xenc:EncryptedKey>` element that contains the `EncryptedKey`.

1112 When referencing the `EncryptedKey` in a message that does not contain the
1113 `<xenc:EncryptedKey>` element, the `<ds:KeyInfo>` element of the referencing construct
1114 MUST contain a `<wsse:SecurityTokenReference>`. The
1115 `<wsse:SecurityTokenReference>` element MUST contain a `<wsse:KeyIdentifier>`
1116 element. The `EncodingType` attribute SHOULD be set to `#Base64Binary`. Other encoding
1117 types MAY be specified if agreed on by all parties. The `wsse11:TokenType` attribute MUST be
1118 set to

1119 `http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-`
1120 `1.1#EncryptedKey`. The identifier for a `<xenc:EncryptedKey>` token is defined as the SHA1
1121 of the raw (pre-base64 encoding) octets specified in the `<xenc:CipherValue>` element of the
1122 referenced `<xenc:EncryptedKey>` token. This value is encoded as indicated in the
1123 `<wsse:KeyIdentifier>` reference. The `<wsse:ValueType>` attribute of
1124 `<wsse:KeyIdentifier>` MUST be set to `http://docs.oasis-open.org/wss/oasis-`
1125 `wss-soap-message-security-1.1#EncryptedKeySHA1`.

8 Signatures

1126

1127 Message producers may want to enable message recipients to determine whether a message
1128 was altered in transit and to verify that the claims in a particular security token apply to the
1129 producer of the message.

1130

1131 Demonstrating knowledge of a confirmation key associated with a token key-claim confirms the
1132 accompanying token claims. Knowledge of a confirmation key may be demonstrated by using
1133 that key to create an XML Signature, for example. The relying party's acceptance of the claims
1134 may depend on its confidence in the token. Multiple tokens may contain a key-claim for a
1135 signature and may be referenced from the signature using a
1136 `<wsse:SecurityTokenReference>`. A key-claim may be an X.509 Certificate token, or a
1137 Kerberos service ticket token to give two examples.

1138

1139 Because of the mutability of some SOAP headers, producers SHOULD NOT use the *Enveloped*
1140 *Signature Transform* defined in XML Signature. Instead, messages SHOULD explicitly include
1141 the elements to be signed. Similarly, producers SHOULD NOT use the *Enveloping Signature*
1142 defined in XML Signature [XMLSIG].

1143

1144 This specification allows for multiple signatures and signature formats to be attached to a
1145 message, each referencing different, even overlapping, parts of the message. This is important
1146 for many distributed applications where messages flow through multiple processing stages. For
1147 example, a producer may submit an order that contains an orderID header. The producer signs
1148 the orderID header and the body of the request (the contents of the order). When this is received
1149 by the order processing sub-system, it may insert a shippingID into the header. The order sub-
1150 system would then sign, at a minimum, the orderID and the shippingID, and possibly the body as
1151 well. Then when this order is processed and shipped by the shipping department, a shippedInfo
1152 header might be appended. The shipping department would sign, at a minimum, the shippedInfo
1153 and the shippingID and possibly the body and forward the message to the billing department for
1154 processing. The billing department can verify the signatures and determine a valid chain of trust
1155 for the order, as well as who authorized each step in the process.

1156

1157 All compliant implementations MUST be able to support the XML Signature standard.

1158

8.1 Algorithms

1159 This specification builds on XML Signature and therefore has the same algorithm requirements as
1160 those specified in the XML Signature specification.

1161

1162

1163

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

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

	Canonicalization	
--	------------------	--

1164
1165
1166

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/

1167
1168
1169
1170
1171
1172
1173
1174
1175
1176

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:

<code><wsse:Security></code> header
[encryption element] [signature element] . .

1177
1178
1179
1180
1181
1182

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:

<code><wsse:Security></code> header
[signature element] [encryption element] . .

1183
1184
1185
1186
1187
1188
1189
1190

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].

1191 There are two problems to be avoided. On the one hand, XML allows documents to be changed
1192 in various ways and still be considered equivalent. For example, duplicate namespace
1193 declarations can be removed or created. As a result, XML tools make these kinds of changes
1194 freely when processing XML. Therefore, it is vital that these equivalent forms match the same
1195 signature.

1196
1197 On the other hand, if the signature simply covers something like `xx:foo`, its meaning may change
1198 if `xx` is redefined. In this case the signature does not prevent tampering. It might be thought that
1199 the problem could be solved by expanding all the values in line. Unfortunately, there are
1200 mechanisms like XPATH which consider `xx="http://example.com/";` to be different from
1201 `yy="http://example.com/";` even though both `xx` and `yy` are bound to the same namespace.
1202 The fundamental difference between the Inclusive and Exclusive Canonicalization is the
1203 namespace declarations which are placed in the output. Inclusive Canonicalization copies all the
1204 declarations that are currently in force, even if they are defined outside of the scope of the
1205 signature. It also copies any `xml:` attributes that are in force, such as `xml:lang` or `xml:base`.
1206 This guarantees that all the declarations you might make use of will be unambiguously specified.
1207 The problem with this is that if the signed XML is moved into another XML document which has
1208 other declarations, the Inclusive Canonicalization will copy them and the signature will be invalid.
1209 This can even happen if you simply add an attribute in a different namespace to the surrounding
1210 context.

1211
1212 Exclusive Canonicalization tries to figure out what namespaces you are actually using and just
1213 copies those. Specifically, it copies the ones that are "visibly used", which means the ones that
1214 are a part of the XML syntax. However, it does not look into attribute values or element content,
1215 so the namespace declarations required to process these are not copied. For example
1216 if you had an attribute like `xx:foo="yy:bar"` it would copy the declaration for `xx`, but not `yy`. (This
1217 can even happen without your knowledge because XML processing tools might add `xsi:type` if
1218 you use a schema subtype.) It also does not copy the `xml:` attributes that are declared outside the
1219 scope of the signature.

1220
1221 Exclusive Canonicalization allows you to create a list of the namespaces that must be declared,
1222 so that it will pick up the declarations for the ones that are not visibly used. The only problem is
1223 that the software doing the signing must know what they are. In a typical SOAP software
1224 environment, the security code will typically be unaware of all the namespaces being used by the
1225 application in the message body that it is signing.

1226
1227 Exclusive Canonicalization is useful when you have a signed XML document that you wish to
1228 insert into other XML documents. A good example is a signed SAML assertion which might be
1229 inserted as a XML Token in the security header of various SOAP messages. The Issuer who
1230 signs the assertion will be aware of the namespaces being used and able to construct the list.
1231 The use of Exclusive Canonicalization will insure the signature verifies correctly every time.
1232 Inclusive Canonicalization is useful in the typical case of signing part or all of the SOAP body in
1233 accordance with this specification. This will insure all the declarations fall under the signature,
1234 even though the code is unaware of what namespaces are being used. At the same time, it is
1235 less likely that the signed data (and signature element) will be inserted in some other XML
1236 document. Even if this is desired, it still may not be feasible for other reasons, for example there
1237 may be Id's with the same value defined in both XML documents.

1238

1239 In other situations it will be necessary to study the requirements of the application and the
1240 detailed operation of the canonicalization methods to determine which is appropriate.
1241 This section is non-normative.

1242 **8.2 Signing Messages**

1243 The `<wsse:Security>` header block MAY be used to carry a signature compliant with the XML
1244 Signature specification within a SOAP Envelope for the purpose of signing one or more elements
1245 in the SOAP Envelope. Multiple signature entries MAY be added into a single SOAP Envelope
1246 within one `<wsse:Security>` header block. Producers SHOULD sign all important elements of
1247 the message, and careful thought must be given to creating a signing policy that requires signing
1248 of parts of the message that might legitimately be altered in transit.

1249
1250 SOAP applications MUST satisfy the following conditions:

- 1251
- 1252 • A compliant implementation MUST be capable of processing the required elements
1253 defined in the XML Signature specification.
- 1254 • To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element
1255 conforming to the XML Signature specification MUST be prepended to the existing
1256 content of the `<wsse:Security>` header block, in order to indicate to the receiver the
1257 correct order of operations. All the `<ds:Reference>` elements contained in the
1258 signature SHOULD refer to a resource within the enclosing SOAP envelope as described
1259 in the XML Signature specification. However, since the SOAP message exchange model
1260 allows intermediate applications to modify the Envelope (add or delete a header block; for
1261 example), XPath filtering does not always result in the same objects after message
1262 delivery. Care should be taken in using XPath filtering so that there is no unintentional
1263 validation failure due to such modifications.
- 1264 • The problem of modification by intermediaries (especially active ones) is applicable to
1265 more than just XPath processing. Digital signatures, because of canonicalization and
1266 digests, present particularly fragile examples of such relationships. If overall message
1267 processing is to remain robust, intermediaries must exercise care that the transformation
1268 algorithms used do not affect the validity of a digitally signed component.
- 1269 • Due to security concerns with namespaces, this specification strongly RECOMMENDS
1270 the use of the "Exclusive XML Canonicalization" algorithm or another canonicalization
1271 algorithm that provides equivalent or greater protection.
- 1272 • For processing efficiency it is RECOMMENDED to have the signature added and then
1273 the security token pre-pended so that a processor can read and cache the token before it
1274 is used.

1275 **8.3 Signing Tokens**

1276 It is often desirable to sign security tokens that are included in a message or even external to the
1277 message. The XML Signature specification provides several common ways for referencing
1278 information to be signed such as URIs, IDs, and XPath, but some token formats may not allow
1279 tokens to be referenced using URIs or IDs and XPaths may be undesirable in some situations.
1280 This specification allows different tokens to have their own unique reference mechanisms which
1281 are specified in their profile as extensions to the `<wsse:SecurityTokenReference>` element.

1282 This element provides a uniform referencing mechanism that is guaranteed to work with all token
1283 formats. Consequently, this specification defines a new reference option for XML Signature: the
1284 STR Dereference Transform.

1285
1286 This transform is specified by the URI #STR-Transform and when applied to a
1287 <wsse:SecurityTokenReference> element it means that the output is the token referenced
1288 by the <wsse:SecurityTokenReference> element not the element itself.

1289
1290 As an overview the processing model is to echo the input to the transform except when a
1291 <wsse:SecurityTokenReference> element is encountered. When one is found, the element
1292 is not echoed, but instead, it is used to locate the token(s) matching the criteria and rules defined
1293 by the <wsse:SecurityTokenReference> element and echo it (them) to the output.
1294 Consequently, the output of the transformation is the resultant sequence representing the input
1295 with any <wsse:SecurityTokenReference> elements replaced by the referenced security
1296 token(s) matched.

1297
1298 The following illustrates an example of this transformation which references a token contained
1299 within the message envelope:

```
1300 ...  
1301 <wsse:SecurityTokenReference wsu:Id="Str1">  
1302   ...  
1303 </wsse:SecurityTokenReference>  
1304 ...  
1305 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
1306   <ds:SignedInfo>  
1307     ...  
1308     <ds:Reference URI="#Str1">  
1309       <ds:Transforms>  
1310         <ds:Transform  
1311           Algorithm="...#STR-Transform">  
1312             <wsse:TransformationParameters>  
1313               <ds:CanonicalizationMethod  
1314                 Algorithm="http://www.w3.org/TR/2001/REC-xml-  
1315 c14n-20010315" />  
1316             </wsse:TransformationParameters>  
1317           </ds:Transform>  
1318           <ds:DigestMethod Algorithm=  
1319             "http://www.w3.org/2000/09/xmldsig#sha1" />  
1320           <ds:DigestValue>...</ds:DigestValue>  
1321         </ds:Reference>  
1322       </ds:SignedInfo>  
1323     <ds:SignatureValue></ds:SignatureValue>  
1324   </ds:Signature>  
1325 ...
```

1326
1327
1328 The following describes the attributes and elements listed in the example above:

1329
1330 */wsse:TransformationParameters*

1331 This element is used to wrap parameters for a transformation allows elements even from
 1332 the XML Signature namespace.
 1333
 1334 */wsse:TransformationParameters/ds:Canonicalization*
 1335 This specifies the canonicalization algorithm to apply to the selected data.
 1336
 1337 */wsse:TransformationParameters/{any}*
 1338 This is an extensibility mechanism to allow different (extensible) parameters to be
 1339 specified in the future. Unrecognized parameters SHOULD cause a fault.
 1340
 1341 */wsse:TransformationParameters/@{any}*
 1342 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 1343 added to the element in the future. Unrecognized attributes SHOULD cause a fault.
 1344
 1345 The following is a detailed specification of the transformation. The algorithm is identified by the
 1346 URI: #STR-Transform.
 1347
 1348 Transform Input:
 1349 • The input is a node set. If the input is an octet stream, then it is automatically parsed; cf.
 1350 XML Digital Signature [XMLSIG].
 1351 Transform Output:
 1352 • The output is an octet steam.
 1353 Syntax:
 1354 • The transform takes a single mandatory parameter, a
 1355 `<ds:CanonicalizationMethod>` element, which is used to serialize the output node
 1356 set. Note, however, that the output may not be strictly in canonical form, per the
 1357 canonicalization algorithm; however, the output is canonical, in the sense that it is
 1358 unambiguous. However, because of syntax requirements in the XML Signature
 1359 definition, this parameter MUST be wrapped in a
 1360 `<wsse:TransformationParameters>` element.
 1361 •
 1362 Processing Rules:
 1363 • Let N be the input node set.
 1364 • Let R be the set of all `<wsse:SecurityTokenReference>` elements in N.
 1365 • For each R_i in R, let D_i be the result of dereferencing R_i .
 1366 • If D_i cannot be determined, then the transform MUST signal a failure.
 1367 • If D_i is an XML security token (e.g., a SAML assertion or a
 1368 `<wsse:BinarySecurityToken>` element), then let R_i' be D_i . Otherwise, D_i is a raw
 1369 binary security token; i.e., an octet stream. In this case, let R_i' be a node set consisting of
 1370 a `<wsse:BinarySecurityToken>` element, utilizing the same namespace prefix as
 1371 the `<wsse:SecurityTokenReference>` element R_i , with no `EncodingType` attribute,
 1372 a `ValueType` attribute identifying the content of the security token, and text content
 1373 consisting of the binary-encoded security token, with no white space.
 1374 • Finally, employ the canonicalization method specified as a parameter to the transform to
 1375 serialize N to produce the octet stream output of this transform; but, in place of any
 1376 dereferenced `<wsse:SecurityTokenReference>` element R_i and its descendants,

1377 process the dereferenced node set Ri' instead. During this step, canonicalization of the
 1378 replacement node set MUST be augmented as follows:
 1379 o Note: A namespace declaration xmlns="" MUST be emitted with every apex
 1380 element that has no namespace node declaring a value for the default
 1381 namespace; cf. XML Decryption Transform.
 1382 Note: Per the processing rules above, any <wsse:SecurityTokenReference>
 1383 element is effectively replaced by the referenced <wsse:BinarySecurityToken>
 1384 element and then the <wsse:BinarySecurityToken> is canonicalized in that
 1385 context. Each <wsse:BinarySecurityToken> needs to be complete in a given
 1386 context, so any necessary namespace declarations that are not present on an ancestor
 1387 element will need to be added to the <wsse:BinarySecurityToken> element prior to
 1388 canonicalization.
 1389
 1390 Signing a <wsse:SecurityTokenReference> (STR) element provides authentication
 1391 and integrity protection of only the STR and not the referenced security token (ST). If
 1392 signing the ST is the intended behavior, the STR Dereference Transform (STRDT) may
 1393 be used which replaces the STR with the ST for digest computation, effectively protecting
 1394 the ST and not the STR. If protecting both the ST and the STR is desired, you may sign
 1395 the STR twice, once using the STRDT and once not using the STRDT.
 1396
 1397 The following table lists the full URI for each URI fragment referred to in the specification.
 1398

URI Fragment	Full URI
#Base64Binary	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
#STR-Transform	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STRTransform
#X509v3	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#X509v3

1399 **8.4 Signature Validation**

1400 The validation of a <ds:Signature> element inside an <wsse:Security> header block
 1401 MUST fail if:
 1402 • the syntax of the content of the element does not conform to this specification, or
 1403 • the validation of the signature contained in the element fails according to the core
 1404 validation of the XML Signature specification [XMLSIG], or
 1405 • the application applying its own validation policy rejects the message for some reason
 1406 (e.g., the signature is created by an untrusted key – verifying the previous two steps only
 1407 performs cryptographic validation of the signature).
 1408
 1409 If the validation of the signature element fails, applications MAY report the failure to the producer
 1410 using the fault codes defined in Section 12 Error Handling.
 1411
 1412 The signature validation shall additionally adhere to the rules defines in signature confirmation
 1413 section below, if the initiator desires signature confirmation:

1414

8.5 Signature Confirmation

1415 In the general model, the initiator uses XML Signature constructs to represent message parts of
1416 the request that were signed. The manifest of signed SOAP elements is contained in the
1417 `<ds:Signature>` element which in turn is placed inside the `<wsse:Security>` header. The
1418 `<ds:Signature>` element of the request contains a `<ds:SignatureValue>`. This element
1419 contains a base64 encoded value representing the actual digital signature. In certain situations it
1420 is desirable that initiator confirms that the message received was generated in response to a
1421 message it initiated in its unaltered form. This helps prevent certain forms of attack. This
1422 specification introduces a `<wsse11:SignatureConfirmation>` element to address this
1423 necessity.

1424

1425 Compliant responder implementations that support signature confirmation, MUST include a
1426 `<wsse11:SignatureConfirmation>` element inside the `<wsse:Security>` header of the
1427 associated response message for every `<ds:Signature>` element that is a direct child of the
1428 `<wsse:Security>` header block in the originating message. The responder MUST include the
1429 contents of the `<ds:SignatureValue>` element of the request signature as the value of the
1430 `@Value` attribute of the `<wsse11:SignatureConfirmation>` element. The
1431 `<wsse11:SignatureConfirmation>` element MUST be included in the message signature of
1432 the associated response message.

1433

1434 If the associated originating signature is received in encrypted form then the corresponding
1435 `<wsse11:SignatureConfirmation>` element SHOULD be encrypted to protect the original
1436 signature and keys.

1437

1438 The schema outline for this element is as follows:

1439

1440

```
<wsse11:SignatureConfirmation wsu:Id="..." Value="..." />
```

1441

1442 */wsse11:SignatureConfirmation*

1443 This element indicates that the responder has processed the signature in the request.

1444 When this element is not present in a response the initiator SHOULD interpret that the
1445 responder is not compliant with this functionality.

1446

1447 */wsse11:SignatureConfirmation/@wsu:Id*

1448 Identifier to be used when referencing this element in the `<ds:SignedInfo>` reference
1449 list of the signature of the associated response message. This attribute MUST be present
1450 so that un-ambiguous references can be made to this

1451 `<wsse11:SignatureConfirmation>` element.

1452

1453 */wsse11:SignatureConfirmation/@Value*

1454 This optional attribute contains the contents of a `<ds:SignatureValue>` copied from
1455 the associated request. If the request was not signed, then this attribute MUST NOT be
1456 present. If this attribute is specified with an empty value, the initiator SHOULD interpret
1457 this as incorrect behavior and process accordingly. When this attribute is not present, the
1458 initiator SHOULD interpret this to mean that the response is based on a request that was
1459 not signed.

1460 8.5.1 Response Generation Rules

1461 Conformant responders MUST include at least one `<wsse1:SignatureConfirmation>`.
1462 element in the `<wsse:Security>` header in any response(s) associated with requests. That is,
1463 the normal messaging patterns are not altered.
1464 For every response message generated, the responder MUST include a
1465 `<wsse1:SignatureConfirmation>` element for every `<ds:Signature>` element it
1466 processed from the original request message. The `Value` attribute MUST be set to the exact
1467 value of the `<ds:SignatureValue>` element of the corresponding `<ds:Signature>` element.
1468 If no `<ds:Signature>` elements are present in the original request message, the responder
1469 MUST include exactly one `<wsse1:SignatureConfirmation>` element. The `Value` attribute
1470 of the `<wsse1:SignatureConfirmation>` element MUST NOT be present. The responder
1471 MUST include all `<wsse1:SignatureConfirmation>` elements in the message signature of
1472 the response message(s). If the `<ds:Signature>` element corresponding to a
1473 `<wsse1:SignatureConfirmation>` element was encrypted in the original request message,
1474 the `<wsse1:SignatureConfirmation>` element SHOULD be encrypted for the recipient of
1475 the response message(s).
1476

1477 8.5.2 Response Processing Rules

1478 The signature validation shall additionally adhere to the following processing guidelines, if the
1479 initiator desires signature confirmation:

- 1480 • If a response message does not contain a `<wsse1:SignatureConfirmation>`
1481 element inside the `<wsse:Security>` header, the initiator SHOULD reject the response
1482 message.
- 1483 • If a response message does contain a `<wsse1:SignatureConfirmation>` element
1484 inside the `<wsse:Security>` header but `@Value` attribute is not present on
1485 `<wsse1:SignatureConfirmation>` element, and the associated request message
1486 did include a `<ds:Signature>` element, the initiator SHOULD reject the response
1487 message.
- 1488 • If a response message does contain a `<wsse1:SignatureConfirmation>` element
1489 inside the `<wsse:Security>` header and the `@Value` attribute is present on the
1490 `<wsse1:SignatureConfirmation>` element, but the associated request did not
1491 include a `<ds:Signature>` element, the initiator SHOULD reject the response
1492 message.
- 1493 • If a response message does contain a `<wsse1:SignatureConfirmation>` element
1494 inside the `<wsse:Security>` header, and the associated request message did include
1495 a `<ds:Signature>` element and the `@Value` attribute is present but does not match the
1496 stored signature value of the associated request message, the initiator SHOULD reject
1497 the response message.
- 1498 • If a response message does not contain a `<wsse1:SignatureConfirmation>`
1499 element inside the `<wsse:Security>` header corresponding to each
1500 `<ds:Signature>` element or if the `@Value` attribute present does not match the stored
1501 signature values of the associated request message, the initiator SHOULD reject the
1502 response message.

1503

8.6 Example

1504

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

1505

1506

1507

1508

1509

1510

1511

1512

1513

1514

1515

1516

1517

1518

1519

1520

1521

1522

1523

1524

1525

1526

1527

1528

1529

1530

1531

1532

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

1549

1550

```
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="...">
  <S11:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken
        ValueType="...#X509v3"
        EncodingType="...#Base64Binary"
        wsu:Id="X509Token">
        MIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm=
            "http://www.w3.org/2001/10/xml-exc-c14n#" />
          <ds:SignatureMethod Algorithm=
            "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <ds:Reference URI="#myBody">
            <ds:Transforms>
              <ds:Transform Algorithm=
                "http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm=
              "http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>EULddytSol...</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
          BL8jdfToEb11/vXcMZNNjPOV...
        </ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#X509Token" />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  <S11:Body wsu:Id="myBody">
    <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
      QQQ
    </tru:StockSymbol>
  </S11:Body>
</S11:Envelope>
```

1551

9 Encryption

1552 This specification allows encryption of any combination of body blocks, header blocks, and any of
1553 these sub-structures by either a common symmetric key shared by the producer and the recipient
1554 or a symmetric key carried in the message in an encrypted form.

1555

1556 In order to allow this flexibility, this specification leverages the XML Encryption standard. This
1557 specification describes how the two elements `<xenc:ReferenceList>` and
1558 `<xenc:EncryptedKey>` listed below and defined in XML Encryption can be used within the
1559 `<wsse:Security>` header block. When a producer or an active intermediary encrypts
1560 portion(s) of a SOAP message using XML Encryption it MUST prepend a sub-element to the
1561 `<wsse:Security>` header block. Furthermore, the encrypting party MUST either prepend the
1562 sub-element to an existing `<wsse:Security>` header block for the intended recipients or create
1563 a new `<wsse:Security>` header block and insert the sub-element. The combined process of
1564 encrypting portion(s) of a message and adding one of these sub-elements is called an encryption
1565 step hereafter. The sub-element MUST contain the information necessary for the recipient to
1566 identify the portions of the message that it is able to decrypt.

1567

1568 This specification additionally defines an element `<wsse11:EncryptedHeader>` for containing
1569 encrypted SOAP header blocks. This specification RECOMMENDS an additional mechanism that
1570 uses this element for encrypting SOAP header blocks that complies with SOAP processing
1571 guidelines while preserving the confidentiality of attributes on the SOAP header blocks.
1572 All compliant implementations MUST be able to support the XML Encryption standard [XMLENC].

1573

9.1 xenc:ReferenceList

1574 The `<xenc:ReferenceList>` element from XML Encryption [XMLENC] MAY be used to
1575 create a manifest of encrypted portion(s), which are expressed as `<xenc:EncryptedData>`
1576 elements within the envelope. An element or element content to be encrypted by this encryption
1577 step MUST be replaced by a corresponding `<xenc:EncryptedData>` according to XML
1578 Encryption. All the `<xenc:EncryptedData>` elements created by this encryption step
1579 SHOULD be listed in `<xenc:DataReference>` elements inside one or more
1580 `<xenc:ReferenceList>` element.

1581

1582 Although in XML Encryption [XMLENC], `<xenc:ReferenceList>` was originally designed to
1583 be used within an `<xenc:EncryptedKey>` element (which implies that all the referenced
1584 `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows
1585 that `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>`
1586 MAY be encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>`
1587 within individual `<xenc:EncryptedData>`.

1588

1589 A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the
1590 producer and the recipient use a shared secret key. The following illustrates the use of this sub-
1591 element:

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

07 November 2005
Page 45 of 76

1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="..." xmlns:xenc="...">
  <S11:Header>
    <wsse:Security>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#bodyID"/>
      </xenc:ReferenceList>
    </wsse:Security>
  </S11:Header>
  <S11:Body>
    <xenc:EncryptedData Id="bodyID">
      <ds:KeyInfo>
        <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </S11:Body>
</S11:Envelope>
```

1613 9.2 xenc:EncryptedKey

1614 When the encryption step involves encrypting elements or element contents within a SOAP
1615 envelope with a symmetric key, which is in turn to be encrypted by the recipient's key and
1616 embedded in the message, <xenc:EncryptedKey> MAY be used for carrying such an
1617 encrypted key. This sub-element MAY contain a manifest, that is, an <xenc:ReferenceList>
1618 element, that lists the portions to be decrypted with this key. The manifest MAY appear outside
1619 the <xenc:EncryptedKey> provided that the corresponding xenc:EncryptedData
1620 elements contain <xenc:KeyInfo> elements that reference the <xenc:EncryptedKey>
1621 element.. An element or element content to be encrypted by this encryption step MUST be
1622 replaced by a corresponding <xenc:EncryptedData> according to XML Encryption. All the
1623 <xenc:EncryptedData> elements created by this encryption step SHOULD be listed in the
1624 <xenc:ReferenceList> element inside this sub-element.

1625
1626 This construct is useful when encryption is done by a randomly generated symmetric key that is
1627 in turn encrypted by the recipient's public key. The following illustrates the use of this element:

1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="..." xmlns:xenc="...">
  <S11:Header>
    <wsse:Security>
      <xenc:EncryptedKey>
        ...
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <ds:X509IssuerSerial>
            <ds:X509IssuerName>
              DC=ACMECorp, DC=com
            </ds:X509IssuerName>
          </ds:X509IssuerSerial>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </xenc:EncryptedKey>
  </S11:Header>
  <S11:Body>
    ...
  </S11:Body>
</S11:Envelope>
```

```

1640         </ds:X509IssuerName>
1641 <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
1642         </ds:X509IssuerSerial>
1643         </wsse:SecurityTokenReference>
1644     </ds:KeyInfo>
1645     ...
1646     </xenc:EncryptedKey>
1647     ...
1648     </wsse:Security>
1649 </S11:Header>
1650 <S11:Body>
1651     <xenc:EncryptedData Id="bodyID">
1652         <xenc:CipherData>
1653             <xenc:CipherValue>...</xenc:CipherValue>
1654         </xenc:CipherData>
1655     </xenc:EncryptedData>
1656 </S11:Body>
1657 </S11:Envelope>

```

1658
1659 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in
1660 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that
1661 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

1662 **9.3 Encrypted Header**

1663 In order to be compliant with SOAP mustUnderstand processing guidelines and to prevent
1664 disclosure of information contained in attributes on a SOAP header block, this specification
1665 introduces an `<wsse11:EncryptedHeader>` element. This element contains exactly one
1666 `<xenc:EncryptedData>` element. This specification RECOMMENDS the use of
1667 `<wsse11:EncryptedHeader>` element for encrypting SOAP header blocks.

1668 **9.4 Processing Rules**

1669 Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the
1670 XML Encryption specification. An encrypted SOAP envelope MUST still be a valid SOAP
1671 envelope. The message creator MUST NOT encrypt the `<S11:Header>`, `<S12:Header>`,
1672 `<S11:Envelope>`, `<S12:Envelope>`, or `<S11:Body>`, `<S12:Body>` elements but MAY
1673 encrypt child elements of either the `<S11:Header>`, `<S12:Header>` and `<S11:Body>` or
1674 `<S12:Body>` elements. Multiple steps of encryption MAY be added into a single
1675 `<wsse:Security>` header block if they are targeted for the same recipient.

1676
1677 When an element or element content inside a SOAP envelope (e.g. the contents of the
1678 `<S11:Body>` or `<S12:Body>` elements) are to be encrypted, it MUST be replaced by an
1679 `<xenc:EncryptedData>`, according to XML Encryption and it SHOULD be referenced from the
1680 `<xenc:ReferenceList>` element created by this encryption step. If the target of reference is
1681 an `EncryptedHeader` as defined in section 9.3 above, see processing rules defined in section
1682 9.5.3 Encryption using `EncryptedHeader` and section 9.5.4 Decryption of `EncryptedHeader`
1683 below.

1684 9.4.1 Encryption

1685 The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1686 this specification are listed below (note that use of `<xenc:ReferenceList>` is
1687 RECOMMENDED. Additionally, if the target of encryption is a SOAP header, processing rules
1688 defined in section 9.5.3 SHOULD be used).

- 1689 • Create a new SOAP envelope.
- 1690 • Create a `<wsse:Security>` header
- 1691 • When an `<xenc:EncryptedKey>` is used, create a `<xenc:EncryptedKey>` sub-
1692 element of the `<wsse:Security>` element. This `<xenc:EncryptedKey>` sub-
1693 element SHOULD contain an `<xenc:ReferenceList>` sub-element, containing a
1694 `<xenc:DataReference>` to each `<xenc:EncryptedData>` element that was
1695 encrypted using that key.
- 1696 • Locate data items to be encrypted, i.e., XML elements, element contents within the target
1697 SOAP envelope.
- 1698 • Encrypt the data items as follows: For each XML element or element content within the
1699 target SOAP envelope, encrypt it according to the processing rules of the XML
1700 Encryption specification [XMLENC]. Each selected original element or element content
1701 MUST be removed and replaced by the resulting `<xenc:EncryptedData>` element.
- 1702 • The optional `<ds:KeyInfo>` element in the `<xenc:EncryptedData>` element MAY
1703 reference another `<ds:KeyInfo>` element. Note that if the encryption is based on an
1704 attached security token, then a `<wsse:SecurityTokenReference>` element SHOULD
1705 be added to the `<ds:KeyInfo>` element to facilitate locating it.
- 1706 • Create an `<xenc:DataReference>` element referencing the generated
1707 `<xenc:EncryptedData>` elements. Add the created `<xenc:DataReference>`
1708 element to the `<xenc:ReferenceList>`.
- 1709 • Copy all non-encrypted data.

1710 9.4.2 Decryption

1711 On receiving a SOAP envelope containing encryption header elements, for each encryption
1712 header element the following general steps should be processed (this section is non-normative.
1713 Additionally, if the target of reference is an `EncryptedHeader`, processing rules as defined in
1714 section 9.5.4 below SHOULD be used):

- 1715
- 1716 1. Identify any decryption keys that are in the recipient's possession, then identifying any
1717 message elements that it is able to decrypt.
- 1718 2. Locate the `<xenc:EncryptedData>` items to be decrypted (possibly using the
1719 `<xenc:ReferenceList>`).
- 1720 3. Decrypt them as follows:
 - 1721 a. For each element in the target SOAP envelope, decrypt it according to the
1722 processing rules of the XML Encryption specification and the processing rules
1723 listed above.
 - 1724 b. If the decryption fails for some reason, applications MAY report the failure to the
1725 producer using the fault code defined in Section 12 Error Handling of this
1726 specification.

1727 c. It is possible for overlapping portions of the SOAP message to be encrypted in
1728 such a way that they are intended to be decrypted by SOAP nodes acting in
1729 different Roles. In this case, the <xenc:ReferenceList> or
1730 <xenc:EncryptedKey> elements identifying these encryption operations will
1731 necessarily appear in different <wsse:Security> headers. Since SOAP does
1732 not provide any means of specifying the order in which different Roles will
1733 process their respective headers, this order is not specified by this specification
1734 and can only be determined by a prior agreement.

1735 9.4.3 Encryption with EncryptedHeader

1736 When it is required that an entire SOAP header block including the top-level element and its
1737 attributes be encrypted, the original header block SHOULD be replaced with a
1738 <wsse11:EncryptedHeader> element. The <wsse11:EncryptedHeader> element MUST
1739 contain the <xenc:EncryptedData> produced by encrypting the header block. A wsu:Id attribute
1740 MAY be added to the <wsse11:EncryptedHeader> element for referencing. If the referencing
1741 <wsse:Security> header block defines a value for the <S12:mustUnderstand> or
1742 <S11:mustUnderstand> attribute, that attribute and associated value MUST be copied to the
1743 <wsse11:EncryptedHeader> element. If the referencing <wsse:Security> header block
1744 defines a value for the S12:role or S11:actor attribute, that attribute and associated value
1745 MUST be copied to the <wsse11:EncryptedHeader> element. If the referencing
1746 <wsse:Security> header block defines a value for the S12:relay attribute, that attribute and
1747 associated value MUST be copied to the <wsse11:EncryptedHeader> element.
1748

1749 Any header block can be replaced with a corresponding <wsse11:EncryptedHeader> header
1750 block. This includes <wsse:Security> header blocks. (In this case, obviously if the encryption
1751 operation is specified in the same security header or in a security header targeted at a node
1752 which is reached after the node targeted by the <wsse11:EncryptedHeader> element, the
1753 decryption will not occur.)
1754

1755 In addition, <wsse11:EncryptedHeader> header blocks can be super-encrypted and replaced
1756 by other <wsse11:EncryptedHeader> header blocks (for wrapping/tunneling scenarios). Any
1757 <wsse:Security> header that encrypts a header block targeted to a particular actor SHOULD
1758 be targeted to that same actor, unless it is a security header.

1759 9.4.4 Processing an EncryptedHeader

1760 The processing model for <wsse11:EncryptedHeader> header blocks is as follows:

- 1761 1. Resolve references to encrypted data specified in the <wsse:Security> header block
1762 targeted at this node. For each reference, perform the following steps.
- 1763 2. If the referenced element does not have a qualified name of
1764 <wsse11:EncryptedHeader> then process as per section 9.5.2 Decryption and stop
1765 the processing steps here.
- 1766 3. Otherwise, extract the <xenc:EncryptedData> element from the
1767 <wsse11:EncryptedHeader> element.

- 1768 4. Decrypt the contents of the `<xenc:EncryptedData>` element as per section 9.5.2
1769 Decryption and replace the `<wsse11:EncryptedHeader>` element with the decrypted
1770 contents.
1771 5. Process the decrypted header block as per SOAP processing guidelines.
1772

1773 Alternatively, a processor may perform a pre-pass over the encryption references in the
1774 `<wsse:Security>` header:

- 1775 1. Resolve references to encrypted data specified in the `<wsse:Security>` header block
1776 targeted at this node. For each reference, perform the following steps.
1777 2. If a referenced element has a qualified name of `<wsse11:EncryptedHeader>` then
1778 replace the `<wsse11:EncryptedHeader>` element with the contained
1779 `<xenc:EncryptedData>` element and if present copy the value of the `wsu:Id` attribute
1780 from the `<wsse11:EncryptedHeader>` element to the `<xenc:EncryptedData>`
1781 element.
1782 3. Process the `<wsse:Security>` header block as normal.
1783

1784 It should be noted that the results of decrypting a `<wsse11:EncryptedHeader>` header block
1785 could be another `<wsse11:EncryptedHeader>` header block. In addition, the result MAY be
1786 targeted at a different role than the role processing the `<wsse11:EncryptedHeader>` header
1787 block.

1788 **9.4.5 Processing the `mustUnderstand` attribute on `EncryptedHeader`**

1789 If the `S11:mustUnderstand` or `S12:mustUnderstand` attribute is specified on the
1790 `<wsse11:EncryptedHeader>` header block, and is true, then the following steps define what it
1791 means to "understand" the `<wsse11:EncryptedHeader>` header block:

- 1792 1. The processor MUST be aware of this element and know how to decrypt and convert into
1793 the original header block. This DOES NOT REQUIRE that the process know that it has
1794 the correct keys or support the indicated algorithms.
1795 2. The processor MUST, after decrypting the encrypted header block, process the
1796 decrypted header block according to the SOAP processing guidelines. The receiver
1797 MUST raise a fault if any content required to adequately process the header block
1798 remains encrypted or if the decrypted SOAP header is not understood and the value of
1799 the `S12:mustUnderstand` or `S11:mustUnderstand` attribute on the decrypted
1800 header block is true. Note that in order to comply with SOAP processing rules in this
1801 case, the processor must roll back any persistent effects of processing the security
1802 header, such as storing a received token.
1803

1804

10 Security Timestamps

1805 It is often important for the recipient to be able to determine the *freshness* of security semantics.
1806 In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.
1807 This specification does not provide a mechanism for synchronizing time. The assumption is that
1808 time is trusted or additional mechanisms, not described here, are employed to prevent replay.
1809 This specification defines and illustrates time references in terms of the `xsd:dateTime` type
1810 defined in XML Schema. It is RECOMMENDED that all time references use this type. All
1811 references MUST be in UTC time. Implementations MUST NOT generate time instants that
1812 specify leap seconds. If, however, other time types are used, then the `ValueType` attribute
1813 (described below) MUST be specified to indicate the data type of the time format. Requestors and
1814 receivers SHOULD NOT rely on other applications supporting time resolution finer than
1815 milliseconds.

1816

1817 The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and
1818 expiration times of the security semantics in a message.

1819

1820 All times MUST be in UTC format as specified by the XML Schema type (`dateTime`). It should be
1821 noted that times support time precision as defined in the XML Schema specification.

1822 The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and
1823 may only be present at most once per header (that is, per SOAP actor/role).

1824

1825 The ordering within the element is as illustrated below. The ordering of elements in the
1826 `<wsu:Timestamp>` element is fixed and MUST be preserved by intermediaries.

1827 The schema outline for the `<wsu:Timestamp>` element is as follows:

1828

```
1829 <wsu:Timestamp wsu:Id="...">  
1830   <wsu:Created ValueType="...">...</wsu:Created>  
1831   <wsu:Expires ValueType="...">...</wsu:Expires>  
1832   ...  
1833 </wsu:Timestamp>
```

1834

1835 The following describes the attributes and elements listed in the schema above:

1836

1837 */wsu:Timestamp*

1838 This is the element for indicating security semantics timestamps.

1839

1840 */wsu:Timestamp/wsui:Created*

1841 This represents the creation time of the security semantics. This element is optional, but
1842 can only be specified once in a `<wsu:Timestamp>` element. Within the SOAP
1843 processing model, creation is the instant that the infoset is serialized for transmission.
1844 The creation time of the message SHOULD NOT differ substantially from its transmission
1845 time. The difference in time should be minimized.

1846

1847 */wsu:Timestamp/wsu:Expires*
1848 This element represents the expiration of the security semantics. This is optional, but
1849 can appear at most once in a `<wsu:Timestamp>` element. Upon expiration, the
1850 requestor asserts that its security semantics are no longer valid. It is strongly
1851 RECOMMENDED that recipients (anyone who processes this message) discard (ignore)
1852 any message whose security semantics have passed their expiration. A Fault code
1853 (`wsu:MessageExpired`) is provided if the recipient wants to inform the requestor that its
1854 security semantics were expired. A service MAY issue a Fault indicating the security
1855 semantics have expired.
1856

1857 */wsu:Timestamp/{any}*
1858 This is an extensibility mechanism to allow additional elements to be added to the
1859 element. Unrecognized elements SHOULD cause a fault.
1860

1861 */wsu:Timestamp/@wsu:Id*
1862 This optional attribute specifies an XML Schema ID that can be used to reference this
1863 element (the timestamp). This is used, for example, to reference the timestamp in a XML
1864 Signature.
1865

1866 */wsu:Timestamp/@{any}*
1867 This is an extensibility mechanism to allow additional attributes to be added to the
1868 element. Unrecognized attributes SHOULD cause a fault.
1869

1870 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
1871 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
1872 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
1873 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
1874 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
1875 judgment of the requestor's likely current clock time by means not described in this specification,
1876 for example an out-of-band clock synchronization protocol. The recipient may also use the
1877 creation time and the delays introduced by intermediate SOAP roles to estimate the degree of
1878 clock skew.
1879

1880 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```
1881 <S11:Envelope xmlns:S11="..." xmlns:wssse="..." xmlns:wsu="...">  
1882   <S11:Header>  
1883     <wsse:Security>  
1884       <wsu:Timestamp wsu:Id="timestamp">  
1885         <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>  
1886         <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>  
1887       </wsu:Timestamp>  
1888       ...  
1889     </wsse:Security>  
1890     ...  
1891   </S11:Header>  
1892   <S11:Body>  
1893     ...  
1894   </S11:Body>
```


1897

11 Extended Example

1898 The following sample message illustrates the use of security tokens, signatures, and encryption.
1899 For this example, the timestamp and the message body are signed prior to encryption. The
1900 decryption transformation is not needed as the signing/encryption order is specified within the
1901 <wsse:Security> header.

1902

1903

1904

1905

1906

1907

1908

1909

1910

1911

1912

1913

1914

1915

1916

1917

1918

1919

1920

1921

1922

1923

1924

1925

1926

1927

1928

1929

1930

1931

1932

1933

1934

1935

1936

1937

1938

1939

1940

1941

1942

1943

1944

1945

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
(003)   <S11:Header>
(004)     <wsse:Security>
(005)       <wsu:Timestamp wsu:Id="T0">
(006)         <wsu:Created>
(007)           2001-09-13T08:42:00Z</wsu:Created>
(008)         </wsu:Timestamp>
(009)
(010)       <wsse:BinarySecurityToken
(011)         ValueType="...#X509v3"
(012)         wsu:Id="X509Token"
(013)         EncodingType="...#Base64Binary">
(014)         MIIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
(015)       </wsse:BinarySecurityToken>
(016)       <xenc:EncryptedKey>
(017)         <xenc:EncryptionMethod Algorithm=
(018)           "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
(019)         <ds:KeyInfo>
(020)           <wsse:SecurityTokenReference>
(021)             <wsse:KeyIdentifier
(022)               EncodingType="...#Base64Binary"
(023)               ValueType="...#X509v3">MIGfMa0GCSq...
(024)             </wsse:KeyIdentifier>
(025)           </ds:KeyInfo>
(026)         <xenc:CipherData>
(027)           <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(028)         </xenc:CipherValue>
(029)         </xenc:CipherData>
(030)         <xenc:ReferenceList>
(031)           <xenc:DataReference URI="#enc1"/>
(032)         </xenc:ReferenceList>
(033)       </xenc:EncryptedKey>
(034)       <ds:Signature>
(035)         <ds:SignedInfo>
(036)           <ds:CanonicalizationMethod
(037)             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
(038)           <ds:SignatureMethod
(039)             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
(040)           <ds:Reference URI="#T0">
(041)             <ds:Transforms>
(042)               <ds:Transform
```

```

1946 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1947 (034) </ds:Transforms>
1948 (035) <ds:DigestMethod
1949 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1950 (036) <ds:DigestValue>LyLsF094hPi4wPU...
1951 (037) </ds:DigestValue>
1952 (038) </ds:Reference>
1953 (039) <ds:Reference URI="#body">
1954 (040) <ds:Transforms>
1955 (041) <ds:Transform
1956 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1957 (042) </ds:Transforms>
1958 (043) <ds:DigestMethod
1959 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1960 (044) <ds:DigestValue>LyLsF094hPi4wPU...
1961 (045) </ds:DigestValue>
1962 (046) </ds:Reference>
1963 (047) </ds:SignedInfo>
1964 (048) <ds:SignatureValue>
1965 (049) HplZkmFZ/2kQLXDJbchm5gK...
1966 (050) </ds:SignatureValue>
1967 (051) <ds:KeyInfo>
1968 (052) <wsse:SecurityTokenReference>
1969 (053) <wsse:Reference URI="#X509Token" />
1970 (054) </wsse:SecurityTokenReference>
1971 (055) </ds:KeyInfo>
1972 (056) </ds:Signature>
1973 (057) </wsse:Security>
1974 (058) </S11:Header>
1975 (059) <S11:Body wsu:Id="body">
1976 (060) <xenc:EncryptedData
1977 Type="http://www.w3.org/2001/04/xmlenc#Element"
1978 wsu:Id="encl">
1979 (061) <xenc:EncryptionMethod
1980 Algorithm="http://www.w3.org/2001/04/xmlenc#triplede-
1981 cbc" />
1982 (062) <xenc:CipherData>
1983 (063) <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1984 (064) </xenc:CipherValue>
1985 (065) </xenc:CipherData>
1986 (066) </xenc:EncryptedData>
1987 (067) </S11:Body>
1988 (068) </S11:Envelope>

```

1989
1990
1991
1992
1993
1994
1995
1996
1997
1998

Let's review some of the key sections of this example:
Lines (003)-(058) contain the SOAP message headers.

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

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

1999 Lines (010)-(012) specify a security token that is associated with the message. In this case, it
2000 specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64
2001 encoding of the certificate.
2002
2003 Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a
2004 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
2005 encrypt the key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the
2006 symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines
2007 (023)-(025) identify the encryption block in the message that uses this symmetric key. In this
2008 case it is only used to encrypt the body (Id="enc1").
2009
2010 Lines (027)-(056) specify the digital signature. In this example, the signature is based on the
2011 X.509 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039)
2012 references the message body.
2013
2014 Lines (048)-(050) indicate the actual signature value – specified in Line (043).
2015
2016 Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the X.509
2017 certificate included in the message. Line (053) provides a URI link to the Lines (010)-(012).
2018 The body of the message is represented by Lines (059)-(067).
2019
2020 Lines (060)-(066) represent the encrypted metadata and form of the body using XML Encryption.
2021 Line (060) indicates that the "element value" is being replaced and identifies this encryption. Line
2022 (061) specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the
2023 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
2024 key as the key references this encryption – Line (024).
2025

2026

12 Error Handling

2027

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

2028

For example:

2029

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

2030

- Invalid or unauthenticated or unauthenticatable security token

2031

- Invalid signature

2032

- Decryption failure

2033

- Referenced security token is unavailable

2034

- Unsupported namespace

2035

2036

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

2037

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

2038

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

2039

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

2040

2041

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

2042

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

2043

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

2044

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

2045

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

2046

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

2047

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

2048

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

2049

2050

The "failure" class of errors are:

2051

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
The message has expired	wsse:MessageExpired

2052

13 Security Considerations

2053

2054

2055

2056

2057

2058

2059

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.

2060

13.1 General Considerations

2061

2062

2063

2064

2065

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*.

2066

2067

2068

2069

2070

2071

2072

2073

2074

2075

2076

2077

2078

2079

2080

- 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)

2081

2082

2083

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.

2084

13.2 Additional Considerations

2085

13.2.1 Replay

2086

2087

2088

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

2089 messages are exchanged via an open network. These can be part of the message or of the
2090 headers defined from other SOAP extensions. Four typical approaches are: Timestamp,
2091 Sequence Number, Expirations and Message Correlation. Signed timestamps MAY be used to
2092 keep track of messages (possibly by caching the most recent timestamp from a specific service)
2093 and detect replays of previous messages. It is RECOMMENDED that timestamps be cached for
2094 a given period of time, as a guideline, a value of five minutes can be used as a minimum to detect
2095 replays, and that timestamps older than that given period of time set be rejected in interactive
2096 scenarios.

2097 **13.2.2 Combining Security Mechanisms**

2098 This specification defines the use of XML Signature and XML Encryption in SOAP headers. As
2099 one of the building blocks for securing SOAP messages, it is intended to be used in conjunction
2100 with other security techniques. Digital signatures need to be understood in the context of other
2101 security mechanisms and possible threats to an entity.

2102 Implementers should also be aware of all the security implications resulting from the use of digital
2103 signatures in general and XML Signature in particular. When building trust into an application
2104 based on a digital signature there are other technologies, such as certificate evaluation, that must
2105 be incorporated, but these are outside the scope of this document.

2106
2107
2108 As described in XML Encryption, the combination of signing and encryption over a common data
2109 item may introduce some cryptographic vulnerability. For example, encrypting digitally signed
2110 data, while leaving the digital signature in the clear, may allow plain text guessing attacks.

2111 **13.2.3 Challenges**

2112 When digital signatures are used for verifying the claims pertaining to the sending entity, the
2113 producer must demonstrate knowledge of the confirmation key. One way to achieve this is to use
2114 a challenge-response type of protocol. Such a protocol is outside the scope of this document.
2115 To this end, the developers can attach timestamps, expirations, and sequences to messages.

2116 **13.2.4 Protecting Security Tokens and Keys**

2117 Implementers should be aware of the possibility of a token substitution attack. In any situation
2118 where a digital signature is verified by reference to a token provided in the message, which
2119 specifies the key, it may be possible for an unscrupulous producer to later claim that a different
2120 token, containing the same key, but different information was intended.

2121 An example of this would be a user who had multiple X.509 certificates issued relating to the
2122 same key pair but with different attributes, constraints or reliance limits. Note that the signature of
2123 the token by its issuing authority does not prevent this attack. Nor can an authority effectively
2124 prevent a different authority from issuing a token over the same key if the user can prove
2125 possession of the secret.

2126
2127 The most straightforward counter to this attack is to insist that the token (or its unique identifying
2128 data) be included under the signature of the producer. If the nature of the application is such that
2129 the contents of the token are irrelevant, assuming it has been issued by a trusted authority, this

2130 attack may be ignored. However because application semantics may change over time, best
2131 practice is to prevent this attack.
2132
2133 Requestors should use digital signatures to sign security tokens that do not include signatures (or
2134 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
2135 RECOMMENDED that all relevant and immutable message content be signed by the producer.
2136 Receivers SHOULD only consider those portions of the document that are covered by the
2137 producer's signature as being subject to the security tokens in the message. Security tokens
2138 appearing in `<wsse:Security>` header elements SHOULD be signed by their issuing authority
2139 so that message receivers can have confidence that the security tokens have not been forged or
2140 altered since their issuance. It is strongly RECOMMENDED that a message producer sign any
2141 `<wsse:SecurityToken>` elements that it is confirming and that are not signed by their issuing
2142 authority.
2143 When a requester provides, within the request, a Public Key to be used to encrypt the response,
2144 it is possible that an attacker in the middle may substitute a different Public Key, thus allowing the
2145 attacker to read the response. The best way to prevent this attack is to bind the encryption key in
2146 some way to the request. One simple way of doing this is to use the same key pair to sign the
2147 request as to encrypt the response. However, if policy requires the use of distinct key pairs for
2148 signing and encryption, then the Public Key provided in the request should be included under the
2149 signature of the request.

2150 **13.2.5 Protecting Timestamps and Ids**

2151 In order to *trust* `wsu:Id` attributes and `<wsu:Timestamp>` elements, they SHOULD be signed
2152 using the mechanisms outlined in this specification. This allows readers of the IDs and
2153 timestamps information to be certain that the IDs and timestamps haven't been forged or altered
2154 in any way. It is strongly RECOMMENDED that IDs and timestamp elements be signed.
2155

2156 **13.2.6 Protecting against removal and modification of XML Elements**

2157 XML Signatures using Shorthand XPointer References (AKA IDREF) protect against the removal
2158 and modification of XML elements; but do not protect the location of the element within the XML
2159 Document.

2160
2161 Whether or not this is a security vulnerability depends on whether the location of the signed data
2162 within its surrounding context has any semantic import. This consideration applies to data carried
2163 in the SOAP Body or the Header.

2164
2165 Of particular concern is the ability to relocate signed data into a SOAP Header block which is
2166 unknown to the receiver and marked `mustUnderstand="false"`. This could have the effect of
2167 causing the receiver to ignore signed data which the sender expected would either be processed
2168 or result in the generation of a MustUnderstand fault.

2169
2170 A similar exploit would involve relocating signed data into a SOAP Header block targeted to a
2171 `S11:actor` or `S12:role` other than that which the sender intended, and which the receiver will not
2172 process.
2173

2174 While these attacks could apply to any portion of the message, their effects are most pernicious
2175 with SOAP header elements which may not always be present, but must be processed whenever
2176 they appear.

2177
2178 In the general case of XML Documents and Signatures, this issue may be resolved by signing the
2179 entire XML Document and/or strict XML Schema specification and enforcement. However,
2180 because elements of the SOAP message, particularly header elements, may be legitimately
2181 modified by SOAP intermediaries, this approach is usually not appropriate. It is RECOMMENDED
2182 that applications signing any part of the SOAP body sign the entire body.

2183
2184 Alternatives countermeasures include (but are not limited to):

- 2185 • References using XPath transforms with Absolute Path expressions with checks
2186 performed by the receiver that the URI and Absolute Path XPath expression evaluate to
2187 the digested nodeset.
- 2188 • A Reference using an XPath transform to include any significant location-dependent
2189 elements and exclude any elements that might legitimately be removed, added, or altered
2190 by intermediaries,
- 2191 • Using only References to elements with location-independent semantics,
- 2192 • Strict policy specification and enforcement regarding which message parts are to be
2193 signed. For example:
 - 2194 ○ Requiring that the entire SOAP Body and all children of SOAP Header be signed,
 - 2195 ○ Requiring that SOAP header elements which are marked
2196 `MustUnderstand="false"` and have signed descendants MUST include the
2197 `MustUnderstand` attribute under the signature.

2198

2199 **13.2.7 Detecting Duplicate Identifiers**

2200 The `<wsse:Security>` processing SHOULD check for duplicate values from among the set of
2201 ID attributes that it is aware of. The `wsse:Security` processing MUST generate a fault if a
2202 duplicate ID value is detected.

2203
2204 This section is non-normative.

2205

14 Interoperability Notes

2206

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.

2207

2208

2209

2210

2211

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

2212

2213

- **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.

2214

2215

2216

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

2217

2218

2219

- **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.

2220

2221

2222

2223

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

2224

2225

2226

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

2227

2228

- **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.

2229

2230

2231

2232

This section is non-normative.

2233

15 Privacy Considerations

2234 In the context of this specification, we are only concerned with potential privacy violation by the
2235 security elements defined here. Privacy of the content of the payload message is out of scope.
2236 Producers or sending applications should be aware that claims, as collected in security tokens,
2237 are typically personal information, and should thus only be sent according to the producer's
2238 privacy policies. Future standards may allow privacy obligations or restrictions to be added to this
2239 data. Unless such standards are used, the producer must ensure by out-of-band means that the
2240 recipient is bound to adhering to all restrictions associated with the data, and the recipient must
2241 similarly ensure by out-of-band means that it has the necessary consent for its intended
2242 processing of the data.

2243

2244 If claim data are visible to intermediaries, then the policies must also allow the release to these
2245 intermediaries. As most personal information cannot be released to arbitrary parties, this will
2246 typically require that the actors are referenced in an identifiable way; such identifiable references
2247 are also typically needed to obtain appropriate encryption keys for the intermediaries.

2248 If intermediaries add claims, they should be guided by their privacy policies just like the original
2249 producers.

2250

2251 Intermediaries may also gain traffic information from a SOAP message exchange, e.g., who
2252 communicates with whom at what time. Producers that use intermediaries should verify that
2253 releasing this traffic information to the chosen intermediaries conforms to their privacy policies.

2254

2255 This section is non-normative.

2256

16References

- 2257 **[GLOSS]** Informational RFC 2828, "Internet Security Glossary," May 2000.
- 2258 **[KERBEROS]** J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993, <http://www.ietf.org/rfc/rfc1510.txt> .
- 2259
- 2260 **[KEYWORDS]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997.
- 2261
- 2262 **[SHA-1]** FIPS PUB 180-1. Secure Hash Standard. U.S. Department of
2263 Commerce / National Institute of Standards and Technology.
2264 <http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>
- 2265 **[SOAP11]** W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.
- 2266 **[SOAP12]** W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging
2267 Framework", 23 June 2003.
- 2268 **[SOAPSEC]** W3C Note, "SOAP Security Extensions: Digital Signature," 06 February
2269 2001.
- 2270 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers
2271 (URI): Generic Syntax," RFC 3986, MIT/LCS, Day Software, Adobe
2272 Systems, January 2005.
- 2273 **[XPath]** W3C Recommendation, "XML Path Language", 16 November 1999
- 2274

2275 The following are non-normative references included for background and related material:

- 2276 **[WS-SECURITY]** "Web Services Security Language", IBM, Microsoft, VeriSign, April 2002.
2277 "WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002.
2278 "WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002.
- 2279 **[XMLC14N]** W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001.
- 2280 **[EXCC14N]** W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8
2281 July 2002.
- 2282 **[XMLENC]** W3C Working Draft, "XML Encryption Syntax and Processing," 04 March
2283 2002.
- 2284 W3C Recommendation, "Decryption Transform for XML Signature", 10 December 2002.
- 2285 **[XML-ns]** W3C Recommendation, "Namespaces in XML," 14 January 1999.
- 2286 **[XMLSCHEMA]** W3C Recommendation, "XML Schema Part 1: Structures," 2 May 2001.
2287 W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.
- 2288 **[XMLSIG]** D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-
2289 Signature Syntax and Processing*, W3C Recommendation, 12 February
2290 2002.

2291	[X509]	S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified Certificates Profile,"
2292		
2293		http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I
2294		
2295	[WSS-SAML]	OASIS Working Draft 06, "Web Services Security SAML Token Profile",
2296		21 February 2003
2297	[WSS-XrML]	OASIS Working Draft 03, "Web Services Security XrML Token Profile",
2298		30 January 2003
2299	[WSS-X509]	OASIS, "Web Services Security X.509 Certificate Token Profile", 19
2300		January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-
2301		200401-wss-x509-token-profile-1.0
2302	[WSSKERBEROS]	OASIS Working Draft 03, "Web Services Security Kerberos Profile", 30
2303		January 2003
2304	[WSSUSERNAME]	OASIS, "Web Services Security UsernameToken Profile" 19 January
2305		2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-
2306		username-token-profile-1.0
2307	[WSS-XCBF]	OASIS Working Draft 1.1, "Web Services Security XCBF Token Profile",
2308		30 March 2003
2309	[XMLID]	W3C Recommendation, "xml:id Version 1.0", 9 September 2005.
2310	[XPOINTER]	"XML Pointer Language (XPointer) Version 1.0, Candidate
2311		Recommendation", DeRose, Maler, Daniel, 11 September 2001.

Appendix A: Acknowledgements

Current Contributors:

Michael	Hu	Actional
Maneesh	Sahu	Actional
Duane	Nickull	Adobe Systems
Gene	Thurston	AmberPoint
Frank	Siebenlist	Argonne National Laboratory
Hal	Lockhart	BEA Systems
Denis	Pilipchuk	BEA Systems
Corinna	Witt	BEA Systems
Steve	Anderson	BMC Software
Rich	Levinson	Computer Associates
Thomas	DeMartini	ContentGuard
Merlin	Hughes	Cybertrust
Dale	Moberg	Cyclone Commerce
Rich	Salz	Datapower
Sam	Wei	EMC
Dana S.	Kaufman	Forum Systems
Toshihiro	Nishimura	Fujitsu
Kefeng	Chen	GeoTrust
Irving	Reid	Hewlett-Packard
Kojiro	Nakayama	Hitachi
Paula	Austel	IBM
Derek	Fu	IBM
Maryann	Hondo	IBM
Kelvin	Lawrence	IBM
Michael	McIntosh	IBM
Anthony	Nadalin	IBM
Nataraj	Nagaratnam	IBM
Bruce	Rich	IBM
Ron	Williams	IBM
Don	Flinn	Individual
Kate	Cherry	Lockheed Martin
Paul	Cotton	Microsoft
Vijay	Gajjala	Microsoft
Martin	Gudgin	Microsoft
Chris	Kaler	Microsoft
Frederick	Hirsch	Nokia
Abbie	Barbir	Nortel
Prateek	Mishra	Oracle
Vamsi	Motukuru	Oracle
Ramana	Turlapi	Oracle
Ben	Hammond	RSA Security

Rob	Philpott	RSA Security
Blake	Dournaee	Sarvega
Sundeeep	Peechu	Sarvega
Coumara	Radja	Sarvega
Pete	Wenzel	SeeBeyond
Manveen	Kaur	Sun Microsystems
Ronald	Monzillo	Sun Microsystems
Jan	Alexander	Systinet
Symon	Chang	TIBCO Software
John	Weiland	US Navy
Hans	Granqvist	VeriSign
Phillip	Hallam-Baker	VeriSign
Hemma	Prafullchandra	VeriSign

2314

Previous Contributors:

Pete	Dapkus	BEA
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard
Xin	Wang	ContentGuard
Shawn	Sharp	Cyclone Commerce
Ganesh	Vaideeswaran	Documentum
Tim	Moses	Entrust
Carolina	Canales-Valenzuela	Ericsson
Tom	Rutt	Fujitsu
Yutaka	Kudo	Hitachi
Jason	Rouault	HP
Bob	Blakley	IBM
Joel	Farrell	IBM
Satoshi	Hada	IBM
Hiroshi	Maruyama	IBM
David	Melgar	IBM
Kent	Tamura	IBM
Wayne	Vicknair	IBM
Phil	Griffin	Individual
Mark	Hayes	Individual
John	Hughes	Individual
Peter	Rostin	Individual
Davanum	Srinivas	Individual
Bob	Morgan	Individual/Internet
Bob	Atkinson	Microsof
Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Giovanni	Della-Libera	Microsoft
Alan	Geller	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
Jeff	Hodges	Neustar
Senthil	Sengodan	Nokia
Lloyd	Burch	Novell
Ed	Reed	Novell
Charles	Knouse	Oblix
Vipin	Samar	Oracle
Jerry	Schwarz	Oracle
Eric	Gravengaard	Reactivity
Andrew	Nash	Reactivity
Stuart	King	Reed Elsevier
Martijn	de Boer	SAP
Jonathan	Tourzan	Sony
Yassir	Elley	Sun
Michael	Nguyen	The IDA of Singapore
Don	Adams	TIBCO
Morten	Jorgensen	Vordel

2315

2316 **Appendix B: Revision History**

Rev	Date	By Whom	What
2317			
2318			This section is non-normative.

2319

Appendix C: Utility Elements and Attributes

2320 These specifications define several elements, attributes, and attribute groups which can be re-
2321 used by other specifications. This appendix provides an overview of these *utility* components. It
2322 should be noted that the detailed descriptions are provided in the specification and this appendix
2323 will reference these sections as well as calling out other aspects not documented in the
2324 specification.

2325 16.1 Identification Attribute

2326 There are many situations where elements within SOAP messages need to be referenced. For
2327 example, when signing a SOAP message, selected elements are included in the signature. XML
2328 Schema Part 2 provides several built-in data types that may be used for identifying and
2329 referencing elements, but their use requires that consumers of the SOAP message either have or
2330 are able to obtain the schemas where the identity or reference mechanisms are defined. In some
2331 circumstances, for example, intermediaries, this can be problematic and not desirable.

2332
2333 Consequently a mechanism is required for identifying and referencing elements, based on the
2334 SOAP foundation, which does not rely upon complete schema knowledge of the context in which
2335 an element is used. This functionality can be integrated into SOAP processors so that elements
2336 can be identified and referred to without dynamic schema discovery and processing.

2337
2338 This specification specifies a namespace-qualified global attribute for identifying an element
2339 which can be applied to any element that either allows arbitrary attributes or specifically allows
2340 this attribute. This is a general purpose mechanism which can be re-used as needed.
2341 A detailed description can be found in Section 4.0 ID References.

2342
2343 This section is non-normative.

2344 16.2 Timestamp Elements

2345 The specification defines XML elements which may be used to express timestamp information
2346 such as creation and expiration. While defined in the context of message security, these
2347 elements can be re-used wherever these sorts of time statements need to be made.

2348
2349 The elements in this specification are defined and illustrated using time references in terms of the
2350 *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this
2351 type for interoperability. It is further RECOMMENDED that all references be in UTC time for
2352 increased interoperability. If, however, other time types are used, then the `valueType` attribute
2353 MUST be specified to indicate the data type of the time format.

2354 The following table provides an overview of these elements:
2355

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.
---------------	---

2356
2357
2358
2359
2360

A detailed description can be found in Section 10.

This section is non-normative.

2361 **16.3 General Schema Types**

2362 The schema for the utility aspects of this specification also defines some general purpose
2363 schema elements. While these elements are defined in this schema for use with this
2364 specification, they are general purpose definitions that may be used by other specifications as
2365 well.

2366
2367
2368

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.

2369
2370
2371

This section is non-normative.

2372

Appendix D: SecurityTokenReference Model

2373 This appendix provides a non-normative overview of the usage and processing models for the
2374 <wsse:SecurityTokenReference> element.

2375

2376 There are several motivations for introducing the <wsse:SecurityTokenReference>
2377 element:

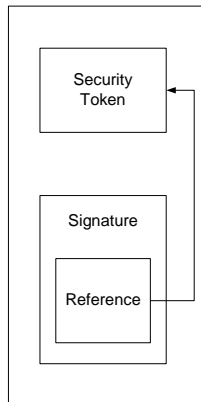
- 2378 • The XML Signature reference mechanisms are focused on "key" references rather than
2379 general token references.
- 2380 • The XML Signature reference mechanisms utilize a fairly closed schema which limits the
2381 extensibility that can be applied.
- 2382 • There are additional types of general reference mechanisms that are needed, but are not
2383 covered by XML Signature.
- 2384 • There are scenarios where a reference may occur outside of an XML Signature and the
2385 XML Signature schema is not appropriate or desired.
- 2386 • The XML Signature references may include aspects (e.g. transforms) that may not apply
2387 to all references.

2388

2389 The following use cases drive the above motivations:

2390

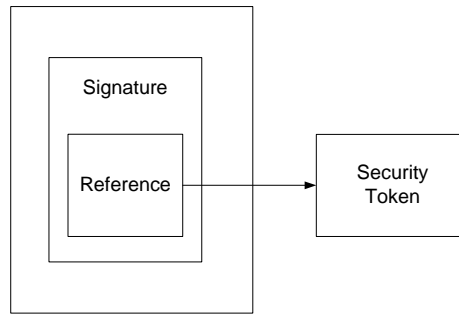
2391 **Local Reference** – A security token, that is included in the message in the <wsse:Security>
2392 header, is associated with an XML Signature. The figure below illustrates this:



2393

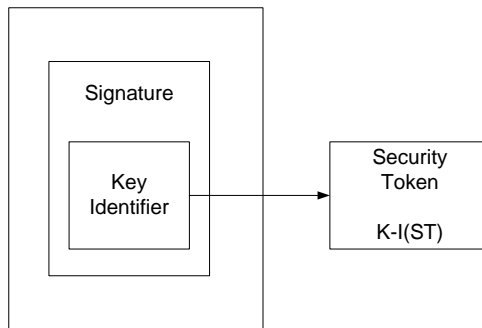
2394
2395
2396
2397

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:



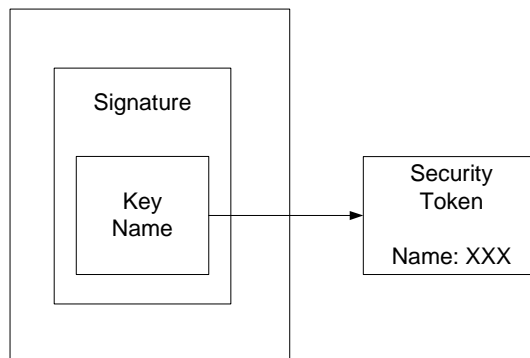
2398
2399
2400
2401

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:



2402
2403
2404
2405

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:

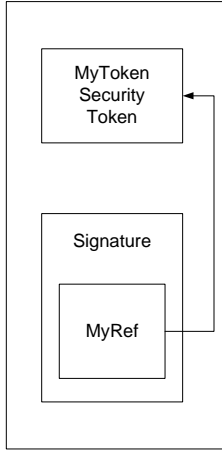


2406
2407
2408

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

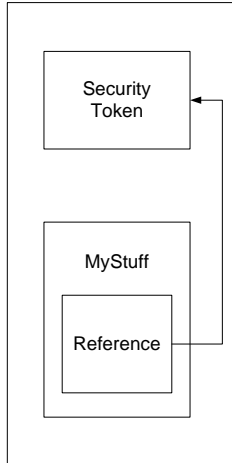
WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2005. All Rights Reserved.

07 November 2005
Page 74 of 76



2409 described above). The figure below illustrates this:
 2410
 2411

Non-Signature References – A message may contain XML that does not represent an XML



2412 signature, but may reference a security token (which may or may not be included in the
 2413 message). The figure below illustrates this:

2414
 2415
 2416
 2417
 2418
 2419
 2420
 2421
 2422
 2423
 2424
 2425
 2426
 2427
 2428
 2429
 2430
 2431

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

The reference may include a `wsse11:TokenType` attribute which provides a "hint" for the type of desired token.

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

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

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

2432 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
2433 references are, by definition, unique by XML. However, other mechanisms such as "principal
2434 name" are not required to be unique and therefore such references may be unique.
2435 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
2436 information about the "key" used in the signature. For token references within signatures, it is
2437 recommended that the `<wsse:SecurityTokenReference>` be placed within the
2438 `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
2439 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
2440 Message Security or its profiles are preferred over the mechanisms in XML Signature.
2441 The following provides additional details on the specific reference mechanisms defined in WSS:
2442 SOAP Message Security:

2443
2444 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
2445 the security token. If only the fragment is specified, then it references the security token within
2446 the document whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to
2447 a [potentially external] security token identified using a URI. There are no implied semantics
2448 around the processing of the URI.
2449

2450 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
2451 by specifying a known value (identifier) for the token, which is determined by applying a special
2452 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
2453 specific security token but requires a profile or token-specific function to be specified. The
2454 `ValueType` attribute defines the type of key identifier and, consequently, identifies the type of
2455 token referenced. The `EncodingType` attribute specifies how the unique value (identifier) is
2456 encoded. For example, a hash value may be encoded using base 64 encoding.
2457

2458 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
2459 specific value that is used to *match* an identity assertion within the security token. This is a
2460 subset match and may result in multiple security tokens that match the specified name. While
2461 XML Signature doesn't imply formatting semantics, WSS: SOAP Message Security recommends
2462 that X.509 names be specified.
2463

2464 It is expected that, where appropriate, profiles define if and how the reference mechanisms map
2465 to the specific token profile. Specifically, the profile should answer the following questions:
2466

- 2467 • What types of references can be used?
- 2468 • How "Key Name" references map (if at all)?
- 2469 • How "Key Identifier" references map (if at all)?
- 2470 • Are there any additional profile or format-specific references?

2471
2472 This section is non-normative.