



1
2

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

6 **OASIS Standard Specification, 1 February 2006**

7 **OASIS identifier:**

8 `wss-v1.1-spec-os-SOAPMessageSecurity`

9 **Location:**

10 `http://docs.oasis-open.org/wss/v1.1/`

11 **Technical Committee:**

12 Web Service Security (WSS)

13 **Chairs:**

14 Kelvin Lawrence, IBM
15 Chris Kaler, Microsoft

16 **Editors:**

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

21 **Abstract:**

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

25
26 This specification also provides a general-purpose mechanism for associating security
27 tokens with message content. No specific type of security token is required, the
28 specification is designed to be extensible (i.e.. support multiple security token formats).
29 For example, a client might provide one format for proof of identity and provide another
30 format for proof that they have a particular business certification.

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 1 of 76

31
32 Additionally, this specification describes how to encode binary security tokens, a
33 framework for XML-based tokens, and how to include opaque encrypted keys. It also
34 includes extensibility mechanisms that can be used to further describe the characteristics
35 of the tokens that are included with a message.

36 **Status:**

37 This is an OASIS Standard document produced by the Web Services Security Technical
38 Committee. It was approved by the OASIS membership on 1 February 2006. Check the
39 current location noted above for possible errata to this document.

40 Technical Committee members should send comments on this specification to the
41 technical Committee's email list. Others should send comments to the Technical
42 Committee by using the "Send A Comment" button on the Technical Committee's web
43 page at www.oasisopen.org/committees/wss.
44

45 For patent disclosure information that may be essential to the implementation of this
46 specification, and any offers of licensing terms, refer to the Intellectual Property Rights
47 section of the OASIS Web Services Security Technical Committee (WSS TC) web page
48 at <http://www.oasis-open.org/committees/wss/ipr.php>. General OASIS IPR information
49 can be found at <http://www.oasis-open.org/who/intellectualproperty.shtml>.

51 Notices

52 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
53 that might be claimed to pertain to the implementation or use of the technology described in this
54 document or the extent to which any license under such rights might or might not be available;
55 neither does it represent that it has made any effort to identify any such rights. Information on
56 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
57 website. Copies of claims of rights made available for publication and any assurances of licenses
58 to be made available, or the result of an attempt made to obtain a general license or permission
59 for the use of such proprietary rights by implementors or users of this specification, can be
60 obtained from the OASIS Executive Director. OASIS invites any interested party to bring to its
61 attention any copyrights, patents or patent applications, or other proprietary rights which may
62 cover technology that may be required to implement this specification. Please address the
63 information to the OASIS Executive Director.

64
65 Copyright (C) OASIS Open 2002-2006. All Rights Reserved.

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

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

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

85
86 OASIS has been notified of intellectual property rights claimed in regard to some or all of the
87 contents of this specification. For more information consult the online list of claimed rights.

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

Table of Contents

91	1	Introduction	7
92	1.1	Goals and Requirements	7
93	1.1.1	Requirements.....	8
94	1.1.2	Non-Goals.....	8
95	2	Notations and Terminology	9
96	2.1	Notational Conventions	9
97	2.2	Namespaces	9
98	2.3	Acronyms and Abbreviations	10
99	2.4	Terminology.....	11
100	2.5	Note on Examples	12
101	3	Message Protection Mechanisms.....	13
102	3.1	Message Security Model.....	13
103	3.2	Message Protection.....	13
104	3.3	Invalid or Missing Claims	14
105	3.4	Example	14
106	4	ID References	17
107	4.1	Id Attribute	17
108	4.2	Id Schema	18
109	5	Security Header	20
110	6	Security Tokens	23
111	6.1	Attaching Security Tokens	23
112	6.1.1	Processing Rules.....	23
113	6.1.2	Subject Confirmation.....	23
114	6.2	User Name Token	23
115	6.2.1	Usernames.....	23
116	6.3	Binary Security Tokens	24
117	6.3.1	Attaching Security Tokens	24
118	6.3.2	Encoding Binary Security Tokens.....	24
119	6.4	XML Tokens	26
120	6.5	EncryptedData Token	26
121	6.6	Identifying and Referencing Security Tokens	26
122	7	Token References.....	27
123	7.1	SecurityTokenReference Element	27
124	7.2	Direct References.....	29

125	7.3 Key Identifiers.....	30
126	7.4 Embedded References	32
127	7.5 ds:KeyInfo	33
128	7.6 Key Names.....	33
129	7.7 Encrypted Key reference.....	34
130	8 Signatures.....	35
131	8.1 Algorithms	35
132	8.2 Signing Messages.....	38
133	8.3 Signing Tokens.....	38
134	8.4 Signature Validation	41
135	8.5 Signature Confirmation	42
136	8.5.1 Response Generation Rules.....	43
137	8.5.2 Response Processing Rules.....	43
138	8.6 Example	44
139	9 Encryption	45
140	9.1 xenc:ReferenceList	45
141	9.2 xenc:EncryptedKey	46
142	9.3 Encrypted Header	47
143	9.4 Processing Rules	47
144	9.4.1 Encryption	48
145	9.4.2 Decryption.....	48
146	9.4.3 Encryption with EncryptedHeader	49
147	9.4.4 Processing an EncryptedHeader.....	49
148	9.4.5 Processing the mustUnderstand attribute on EncryptedHeader	50
149	10 Security Timestamps	51
150	11 Extended Example.....	54
151	12 Error Handling.....	57
152	13 Security Considerations.....	59
153	13.1 General Considerations	59
154	13.2 Additional Considerations	59
155	13.2.1 Replay.....	59
156	13.2.2 Combining Security Mechanisms	60
157	13.2.3 Challenges.....	60
158	13.2.4 Protecting Security Tokens and Keys.....	60
159	13.2.5 Protecting Timestamps and Ids	61
160	13.2.6 Protecting against removal and modification of XML Elements	61
161	13.2.7 Detecting Duplicate Identifiers	62
162	14 Interoperability Notes.....	63

163	15	Privacy Considerations	64
164	16	References.....	65
165		Appendix A: Acknowledgements	67
166		Appendix B: Revision History	70
167		Appendix C: Utility Elements and Attributes.....	71
168	16.1	Identification Attribute.....	71
169	16.2	Timestamp Elements	71
170	16.3	General Schema Types	72
171		Appendix D: SecurityTokenReference Model	73
172			

173 1 Introduction

174 This OASIS specification is the result of significant new work by the WSS Technical Committee
175 and supersedes the input submissions, Web Service Security (WS-Security) Version 1.0 April 5,
176 2002 and Web Services Security Addendum Version 1.0 August 18, 2002.

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

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

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

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

199 1.1 Goals and Requirements

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

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

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

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

214

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

216 **1.1.1 Requirements**

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

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

224 **1.1.2 Non-Goals**

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

- 226
- 227 • Establishing a security context or authentication mechanisms.
- 228 • Key derivation.
- 229 • Advertisement and exchange of security policy.
- 230 • How trust is established or determined.
- 231 • Non-repudiation.
- 232

233

2 Notations and Terminology

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

2.1 Notational Conventions

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

239

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

243

244 When describing concrete XML schemas, this specification uses a convention where each
245 member of an element's [children] or [attributes] property is described using an XPath-like
246 notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence
247 of an element wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute
248 wildcard (<xs:anyAttribute/>).

249

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

2.2 Namespaces

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

254

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

258

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

261

```
262 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-  
263 secext-1.0.xsd
```

264

```
264 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-  
265 utility-1.0.xsd
```

266

```
266 http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd
```

267

268 This specification is designed to work with the general SOAP [SOAP11, SOAP12] message
269 structure and message processing model, and should be applicable to any version of SOAP. The
270 current SOAP 1.1 namespace URI is used herein to provide detailed examples, but there is no
271 intention to limit the applicability of this specification to a single version of SOAP.

272

273 The namespaces used in this document are shown in the following table (note that for brevity, the
 274 examples use the prefixes listed below but do not include the URIs – those listed below are
 275 assumed).
 276

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#

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

283 **2.3 Acronyms and Abbreviations**

284 The following (non-normative) table defines acronyms and abbreviations for this document.
 285

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

286 **2.4 Terminology**

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

288

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

291

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

294

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

297

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

299

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

307

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

314

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

316

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

319

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

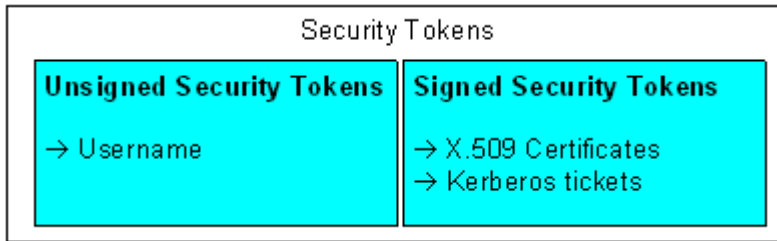
322

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

325

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

327



328
329

330 **Signed Security Token** – A *signed security token* is a security token that is asserted and
331 cryptographically signed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket).
332

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

335 **2.5 Note on Examples**

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

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

348 **3 Message Protection Mechanisms**

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

351

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

357

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

359 **3.1 Message Security Model**

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

362

363 Security tokens assert claims and can be used to assert the binding between authentication
364 secrets or keys and security identities. An authority can vouch for or endorse the claims in a
365 security token by using its key to sign or encrypt (it is recommended to use a keyed encryption)
366 the security token thereby enabling the authentication of the claims in the token. An X.509 [X509]
367 certificate, claiming the binding between one's identity and public key, is an example of a signed
368 security token endorsed by the certificate authority. In the absence of endorsement by a third
369 party, the recipient of a security token may choose to accept the claims made in the token based
370 on its trust of the producer of the containing message.

371

372 Signatures are used to verify message origin and integrity. Signatures are also used by message
373 producers to demonstrate knowledge of the key, typically from a third party, used to confirm the
374 claims in a security token and thus to bind their identity (and any other claims occurring in the
375 security token) to the messages they create.

376

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

379

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

383 **3.2 Message Protection**

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

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

401 **3.3 Invalid or Missing Claims**

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

409 **3.4 Example**

410 The following example illustrates the use of a custom security token and associated signature.
411 The token contains base64 encoded binary data conveying a symmetric key which, we assume,
412 can be properly authenticated by the recipient. The message producer uses the symmetric key
413 with an HMAC signing algorithm to sign the message. The message receiver uses its knowledge
414 of the shared secret to repeat the HMAC key calculation which it uses to validate the signature
415 and in the process confirm that the message was authored by the claimed user identity.
416

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

```

434         Algorithm=
435         "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
436     (012)     <ds:Reference URI="#MsgBody">
437     (013)         <ds:DigestMethod
438                 Algorithm=
439                 "http://www.w3.org/2000/09/xmldsig#sha1"/>
440     (014)         <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
441     (015)         </ds:Reference>
442     (016)     </ds:SignedInfo>
443     (017)     <ds:SignatureValue>Djbchm5gK...</ds:SignatureValue>
444     (018)     <ds:KeyInfo>
445     (019)         <wsse:SecurityTokenReference>
446     (020)             <wsse:Reference URI="#MyID"/>
447     (021)         </wsse:SecurityTokenReference>
448     (022)     </ds:KeyInfo>
449     (023)     </ds:Signature>
450     (024) </wsse:Security>
451     (025) </S11:Header>
452     (026) <S11:Body wsu:Id="MsgBody">
453     (027)     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
454             QQQ
455     </tru:StockSymbol>
456     (028) </S11:Body>
457     (029) </S11:Envelope>

```

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

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

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

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

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

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

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

482

483 Lines (018) to (022) provides information, partial or complete, as to where to find the security
484 token associated with this signature. Specifically, lines (019) to (021) indicate that the security
485 token can be found at (pulled from) the specified URL.
486
487 Lines (026) to (028) contain the body (payload) of the SOAP message.
488

489

4 ID References

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

500

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

506

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

510

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

516

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

521

4.1 Id Attribute

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

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 17 of 76

530

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

535

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

539

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

543 4.2 Id Schema

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

547 The syntax for this attribute is as follows:

548

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

550

551 The following describes the attribute illustrated above:

552 `.../@wsu:Id`

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

555

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

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

560

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

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

564

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

567

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

570

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

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

579

5 Security Header

580 The `<wsse:Security>` header block provides a mechanism for attaching security-related
581 information targeted at a specific recipient in the form of a SOAP actor/role. This may be either
582 the ultimate recipient of the message or an intermediary. Consequently, elements of this type
583 may be present multiple times in a SOAP message. An active intermediary on the message path
584 MAY add one or more new sub-elements to an existing `<wsse:Security>` header block if they
585 are targeted for its SOAP node or it MAY add one or more new headers for additional targets.
586

587 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted
588 for separate recipients. A message MUST NOT have multiple `<wsse:Security>` header blocks
589 targeted (whether explicitly or implicitly) at the same recipient. However, only one
590 `<wsse:Security>` header block MAY omit the `S11:actor` or `S12:role` attributes. Two
591 `<wsse:Security>` header blocks MUST NOT have the same value for `S11:actor` or
592 `S12:role`. Message security information targeted for different recipients MUST appear in
593 different `<wsse:Security>` header blocks. This is due to potential processing order issues
594 (e.g. due to possible header re-ordering). The `<wsse:Security>` header block without a
595 specified `S11:actor` or `S12:role` MAY be processed by anyone, but MUST NOT be removed
596 prior to the final destination or endpoint.
597

598 As elements are added to a `<wsse:Security>` header block, they SHOULD be prepended to
599 the existing elements. As such, the `<wsse:Security>` header block represents the signing and
600 encryption steps the message producer took to create the message. This prepending rule
601 ensures that the receiving application can process sub-elements in the order they appear in the
602 `<wsse:Security>` header block, because there will be no forward dependency among the sub-
603 elements. Note that this specification does not impose any specific order of processing the sub-
604 elements. The receiving application can use whatever order is required.
605

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

610

611

612

613

614

615

616

617

618

619

620

621

622

```
<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:

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 20 of 76

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

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

677 **6 Security Tokens**

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

680 **6.1 Attaching Security Tokens**

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

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

687 **6.1.1 Processing Rules**

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

692 **6.1.2 Subject Confirmation**

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

696 **6.2 User Name Token**

697 **6.2.1 Usernames**

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

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

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

707
708 */wsse:UsernameToken*

709 This element is used to represent a claimed identity.

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

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

```

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

```

748 6.3 Binary Security Tokens

749 6.3.1 Attaching Security Tokens

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

753 6.3.2 Encoding Binary Security Tokens

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

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

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

```
763  
764 <wsse:BinarySecurityToken wsu:Id=...  
765                               EncodingType=...  
766                               ValueType=.../>
```

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

769 */wsse:BinarySecurityToken*

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

771

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

773 An optional string label for this security token.

774

775 */wsse:BinarySecurityToken/@ValueType*

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

781

782 */wsse:BinarySecurityToken/@EncodingType*

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

789

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

790

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

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

794

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

797 **6.4 XML Tokens**

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

800 **6.5 EncryptedData Token**

801 In certain cases it is desirable that the token included in the `<wsse:Security>` header be
802 encrypted for the recipient processing role. In such a case the `<xenc:EncryptedData>`
803 element MAY be used to contain a security token and included in the `<wsse:Security>`
804 header. That is this specification defines the usage of `<xenc:EncryptedData>` to encrypt
805 security tokens contained in `<wsse:Security>` header.
806

807 It should be noted that token references are not made to the `<xenc:EncryptedData>` element,
808 but instead to the token represented by the clear-text, once the `<xenc:EncryptedData>`
809 element has been processed (decrypted). Such references utilize the token profile for the
810 contained token. i.e., `<xenc:EncryptedData>` SHOULD NOT include an XML ID for
811 referencing the contained security token.
812

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

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

817 **6.6 Identifying and Referencing Security Tokens**

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

823

7 Token References

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

7.1 SecurityTokenReference Element

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

831

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

836

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

840 The following illustrates the syntax of this element:

841

```
842 <wsse:SecurityTokenReference wsu:Id="...", wsse11:TokenType="...",  
843 wsse:Usage="...", wsse:Usage="...">  
844 </wsse:SecurityTokenReference>
```

845

846 The following describes the elements defined above:

847

848 */wsse:SecurityTokenReference*

849 This element provides a reference to a security token.

850

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

852 A string label for this security token reference which names the reference. This attribute
853 does not indicate the ID of what is being referenced, that **SHOULD** be done using a

854 fragment URI in a `<wsse:Reference>` element within the

855 `<wsse:SecurityTokenReference>` element.

856

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

858 This optional attribute is used to identify, by URI, the type of the referenced token.

859 This specification recommends that token specific profiles define appropriate token type

860 identifying URI values, and that these same profiles require that these values be

861 specified in the profile defined reference forms.

862

863 When a `wss11:TokenType` attribute is specified in conjunction with a
 864 `wsse:KeyIdentifier/@ValueType` attribute or a `wsse:Reference/@ValueType`
 865 attribute that indicates the type of the referenced token, the security token type identified
 866 by the `wss11:TokenType` attribute MUST be consistent with the security token type
 867 identified by the `wsse:ValueType` attribute.
 868

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

869
 870 */wsse:SecurityTokenReference/@wsse:Usage*
 871 This optional attribute is used to type the usage of the
 872 `<wsse:SecurityTokenReference>`. Usages are specified using URIs and multiple
 873 usages MAY be specified using XML list semantics. No usages are defined by this
 874 specification.
 875
 876 */wsse:SecurityTokenReference/{any}*
 877 This is an extensibility mechanism to allow different (extensible) types of security
 878 references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
 879 fault.
 880
 881 */wsse:SecurityTokenReference/@{any}*
 882 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 883 added to the header. Unrecognized attributes SHOULD cause a fault.
 884
 885 All compliant implementations MUST be able to process a
 886 `<wsse:SecurityTokenReference>` element.
 887
 888 This element can also be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to
 889 retrieve the key information from a security token placed somewhere else. In particular, it is
 890 RECOMMENDED, when using XML Signature and XML Encryption, that a
 891 `<wsse:SecurityTokenReference>` element be placed inside a `<ds:KeyInfo>` to reference
 892 the security token used for the signature or encryption.
 893
 894 There are several challenges that implementations face when trying to interoperate. Processing
 895 the IDs and references requires the recipient to *understand* the schema. This may be an
 896 expensive task and in the general case impossible as there is no way to know the "schema
 897 location" for a specific namespace URI. As well, the primary goal of a reference is to uniquely
 898 identify the desired token. ID references are, by definition, unique by XML. However, other
 899 mechanisms such as "principal name" are not required to be unique and therefore such
 900 references may be not unique.
 901

902 This specification allows for the use of multiple reference mechanisms within a single
903 `<wsse:SecurityTokenReference>`. When multiple references are present in a given
904 `<wsse:SecurityTokenReference>`, they **MUST** resolve to a single token in common.
905 Specific token profiles **SHOULD** define the reference mechanisms to be used.
906

907 The following list provides a list of the specific reference mechanisms defined in WSS: SOAP
908 Message Security in preferred order (i.e., most specific to least specific):
909

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

919 **7.2 Direct References**

920 The `<wsse:Reference>` element provides an extensible mechanism for directly referencing
921 security tokens using URIs.
922

923 The following illustrates the syntax of this element:

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

928
929 The following describes the elements defined above:

930 */wsse:SecurityTokenReference/wsse:Reference*

932 This element is used to identify an abstract URI location for locating a security token.
933

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

935 This optional attribute specifies an abstract URI for a security token. If a fragment is
936 specified, then it indicates the local ID of the security token being referenced. The URI
937 **MUST** identify a security token. The URI **MUST NOT** identify a

938 `wsse:SecurityTokenReference` element, a `wsse:Embedded` element, a
939 `wsse:Reference` element, or a `wsse:KeyIdentifier` element.
940

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

942 This optional attribute specifies a URI that is used to identify the *type* of token being
943 referenced. This specification does not define any processing rules around the usage of
944 this attribute, however, specifications for individual token types **MAY** define specific
945 processing rules and semantics around the value of the URI and its interpretation. If this
946 attribute is not present, the URI **MUST** be processed as a normal URI.
947

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

954
955 `/wsse:SecurityTokenReference/wsse:Reference/{any}`
956 This is an extensibility mechanism to allow different (extensible) types of security
957 references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
958 fault.

959
960 `/wsse:SecurityTokenReference/wsse:Reference/@{any}`
961 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
962 added to the header. Unrecognized attributes SHOULD cause a fault.

963
964 The following illustrates the use of this element:

```
965 <wsse:SecurityTokenReference  
966     xmlns:wsse="...">  
967   <wsse:Reference  
968     URI="http://www.fabrikaml23.com/tokens/Zoe"/>  
969 </wsse:SecurityTokenReference>
```

971 7.3 Key Identifiers

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

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

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

988 The following is an overview of the syntax:

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

```

994     ...
995     </wsse:KeyIdentifier>
996 </wsse:SecurityTokenReference>
997

```

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

999
1000 */wsse:SecurityTokenReference/wsse:KeyIdentifier*

1001 This element is used to include a binary-encoded key identifier.

1002

1003 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@wsu:Id*

1004 An optional string label for this identifier.

1005

1006 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@ValueType*

1007 The optional `ValueType` attribute is used to indicate the type of `KeyIdentifier` being used.

1008 This specification defines one `ValueType` that can be applied to all token types. Each specific

1009 token profile specifies the `KeyIdentifier` types that may be used to refer to tokens of that

1010 type. It also specifies the critical semantics of the identifier, such as whether the

1011 `KeyIdentifier` is unique to the key or the token. If no value is specified then the key identifier

1012 will be interpreted in an application-specific manner. This URI fragment is relative to a base URI

1013 as indicated in the table below.

1014

URI	Description
http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbPrintSHA1	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.
http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKeySHA1	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.
--	---

1015
1016
1017
1018
1019
1020
1021
1022
1023

/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

1024
1025
1026
1027

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

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

1028 **7.4 Embedded References**

1029 In some cases a reference may be to an embedded token (as opposed to a pointer to a token
1030 that resides elsewhere). To do this, the `<wsse:Embedded>` element is specified within a
1031 `<wsse:SecurityTokenReference>` element. The `<wsse:Embedded>` element is only
1032 allowed inside a `<wsse:SecurityTokenReference>` element.

1033 The following is an overview of the syntax:

1034
1035
1036
1037
1038
1039
1040

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

1041 The following describes the attributes and elements listed in the example above:

1042
1043
1044
1045
1046

/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).

1047
1048
1049
1050

/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.

1051
1052
1053

/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.

1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076

/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>
```

1077

7.5 ds:KeyInfo

1078
1079
1080
1081
1082
1083

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.

1084
1085
1086
1087
1088

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>
```

1089

7.6 Key Names

1090
1091
1092
1093
1094
1095
1096

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
```

1097 7.7 Encrypted Key reference

1098 In certain cases, an `<xenc:EncryptedKey>` element MAY be used to carry key material
1099 encrypted for the recipient's key. This key material is henceforth referred to as `EncryptedKey`.

1100
1101 The `EncryptedKey` MAY be used to perform other cryptographic operations within the same
1102 message, such as signatures. The `EncryptedKey` MAY also be used for performing
1103 cryptographic operations in subsequent messages exchanged by the two parties. Two
1104 mechanisms are defined for referencing the `EncryptedKey`.

1105
1106 When referencing the `EncryptedKey` within the same message that contains the
1107 `<xenc:EncryptedKey>` element, the `<ds:KeyInfo>` element of the referencing construct
1108 MUST contain a `<wsse:SecurityTokenReference>`. The
1109 `<wsse:SecurityTokenReference>` element MUST contain a `<wsse:Reference>` element.

1110
1111 The URI attribute value of the `<wsse:Reference>` element MUST be set to the value of the ID
1112 attribute of the referenced `<xenc:EncryptedKey>` element that contains the `EncryptedKey`.
1113 When referencing the `EncryptedKey` in a message that does not contain the
1114 `<xenc:EncryptedKey>` element, the `<ds:KeyInfo>` element of the referencing construct
1115 MUST contain a `<wsse:SecurityTokenReference>`. The
1116 `<wsse:SecurityTokenReference>` element MUST contain a `<wsse:KeyIdentifier>`
1117 element. The `EncodingType` attribute SHOULD be set to `#Base64Binary`. Other encoding
1118 types MAY be specified if agreed on by all parties. The `wsse11:TokenType` attribute MUST be
1119 set to
1120 `http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-`
1121 `1.1#EncryptedKey`. The identifier for a `<xenc:EncryptedKey>` token is defined as the SHA1
1122 of the raw (pre-base64 encoding) octets specified in the `<xenc:CipherValue>` element of the
1123 referenced `<xenc:EncryptedKey>` token. This value is encoded as indicated in the
1124 `<wsse:KeyIdentifier>` reference. The `<wsse:ValueType>` attribute of
1125 `<wsse:KeyIdentifier>` MUST be set to `http://docs.oasis-open.org/wss/oasis-`
1126 `wss-soap-message-security-1.1#EncryptedKeySHA1`.

1127

8 Signatures

1128 Message producers may want to enable message recipients to determine whether a message
1129 was altered in transit and to verify that the claims in a particular security token apply to the
1130 producer of the message.

1131

1132 Demonstrating knowledge of a confirmation key associated with a token key-claim confirms the
1133 accompanying token claims. Knowledge of a confirmation key may be demonstrated by using
1134 that key to create an XML Signature, for example. The relying party's acceptance of the claims
1135 may depend on its confidence in the token. Multiple tokens may contain a key-claim for a
1136 signature and may be referenced from the signature using a
1137 `<wsse:SecurityTokenReference>`. A key-claim may be an X.509 Certificate token, or a
1138 Kerberos service ticket token to give two examples.

1139

1140 Because of the mutability of some SOAP headers, producers SHOULD NOT use the *Enveloped*
1141 *Signature Transform* defined in XML Signature. Instead, messages SHOULD explicitly include
1142 the elements to be signed. Similarly, producers SHOULD NOT use the *Enveloping Signature*
1143 defined in XML Signature [XMLSIG].

1144

1145 This specification allows for multiple signatures and signature formats to be attached to a
1146 message, each referencing different, even overlapping, parts of the message. This is important
1147 for many distributed applications where messages flow through multiple processing stages. For
1148 example, a producer may submit an order that contains an orderID header. The producer signs
1149 the orderID header and the body of the request (the contents of the order). When this is received
1150 by the order processing sub-system, it may insert a shippingID into the header. The order sub-
1151 system would then sign, at a minimum, the orderID and the shippingID, and possibly the body as
1152 well. Then when this order is processed and shipped by the shipping department, a shippedInfo
1153 header might be appended. The shipping department would sign, at a minimum, the shippedInfo
1154 and the shippingID and possibly the body and forward the message to the billing department for
1155 processing. The billing department can verify the signatures and determine a valid chain of trust
1156 for the order, as well as who authorized each step in the process.

1157

1158 All compliant implementations MUST be able to support the XML Signature standard.

1159

8.1 Algorithms

1160 This specification builds on XML Signature and therefore has the same algorithm requirements as
1161 those specified in the XML Signature specification.

1162 The following table outlines additional algorithms that are strongly RECOMMENDED by this
1163 specification:

1164

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

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 35 of 76

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

1165
1166
1167

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/

1168
1169
1170
1171
1172
1173
1174
1175
1176
1177

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] . .

1178
1179
1180
1181
1182
1183

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] . .

1184
1185
1186
1187
1188
1189
1190
1191

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].

1192 There are two problems to be avoided. On the one hand, XML allows documents to be changed
1193 in various ways and still be considered equivalent. For example, duplicate namespace
1194 declarations can be removed or created. As a result, XML tools make these kinds of changes
1195 freely when processing XML. Therefore, it is vital that these equivalent forms match the same
1196 signature.

1197
1198 On the other hand, if the signature simply covers something like `xx:foo`, its meaning may change
1199 if `xx` is redefined. In this case the signature does not prevent tampering. It might be thought that
1200 the problem could be solved by expanding all the values in line. Unfortunately, there are
1201 mechanisms like XPATH which consider `xx="http://example.com/"`; to be different from
1202 `yy="http://example.com/"`; even though both `xx` and `yy` are bound to the same namespace.
1203 The fundamental difference between the Inclusive and Exclusive Canonicalization is the
1204 namespace declarations which are placed in the output. Inclusive Canonicalization copies all the
1205 declarations that are currently in force, even if they are defined outside of the scope of the
1206 signature. It also copies any `xml:` attributes that are in force, such as `xml:lang` or `xml:base`.
1207 This guarantees that all the declarations you might make use of will be unambiguously specified.
1208 The problem with this is that if the signed XML is moved into another XML document which has
1209 other declarations, the Inclusive Canonicalization will copy them and the signature will be invalid.
1210 This can even happen if you simply add an attribute in a different namespace to the surrounding
1211 context.

1212
1213 Exclusive Canonicalization tries to figure out what namespaces you are actually using and just
1214 copies those. Specifically, it copies the ones that are "visibly used", which means the ones that
1215 are a part of the XML syntax. However, it does not look into attribute values or element content,
1216 so the namespace declarations required to process these are not copied. For example
1217 if you had an attribute like `xx:foo="yy:bar"` it would copy the declaration for `xx`, but not `yy`. (This
1218 can even happen without your knowledge because XML processing tools might add `xsi:type` if
1219 you use a schema subtype.) It also does not copy the `xml:` attributes that are declared outside the
1220 scope of the signature.

1221
1222 Exclusive Canonicalization allows you to create a list of the namespaces that must be declared,
1223 so that it will pick up the declarations for the ones that are not visibly used. The only problem is
1224 that the software doing the signing must know what they are. In a typical SOAP software
1225 environment, the security code will typically be unaware of all the namespaces being used by the
1226 application in the message body that it is signing.

1227
1228 Exclusive Canonicalization is useful when you have a signed XML document that you wish to
1229 insert into other XML documents. A good example is a signed SAML assertion which might be
1230 inserted as a XML Token in the security header of various SOAP messages. The Issuer who
1231 signs the assertion will be aware of the namespaces being used and able to construct the list.
1232 The use of Exclusive Canonicalization will insure the signature verifies correctly every time.
1233 Inclusive Canonicalization is useful in the typical case of signing part or all of the SOAP body in
1234 accordance with this specification. This will insure all the declarations fall under the signature,
1235 even though the code is unaware of what namespaces are being used. At the same time, it is
1236 less likely that the signed data (and signature element) will be inserted in some other XML
1237 document. Even if this is desired, it still may not be feasible for other reasons, for example there
1238 may be Id's with the same value defined in both XML documents.

1239

1240 In other situations it will be necessary to study the requirements of the application and the
1241 detailed operation of the canonicalization methods to determine which is appropriate.
1242 This section is non-normative.

1243 8.2 Signing Messages

1244 The `<wsse:Security>` header block MAY be used to carry a signature compliant with the XML
1245 Signature specification within a SOAP Envelope for the purpose of signing one or more elements
1246 in the SOAP Envelope. Multiple signature entries MAY be added into a single SOAP Envelope
1247 within one `<wsse:Security>` header block. Producers SHOULD sign all important elements of
1248 the message, and careful thought must be given to creating a signing policy that requires signing
1249 of parts of the message that might legitimately be altered in transit.

1250
1251 SOAP applications MUST satisfy the following conditions:
1252

- 1253 • A compliant implementation MUST be capable of processing the required elements
1254 defined in the XML Signature specification.
- 1255 • To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element
1256 conforming to the XML Signature specification MUST be prepended to the existing
1257 content of the `<wsse:Security>` header block, in order to indicate to the receiver the
1258 correct order of operations. All the `<ds:Reference>` elements contained in the
1259 signature SHOULD refer to a resource within the enclosing SOAP envelope as described
1260 in the XML Signature specification. However, since the SOAP message exchange model
1261 allows intermediate applications to modify the Envelope (add or delete a header block; for
1262 example), XPath filtering does not always result in the same objects after message
1263 delivery. Care should be taken in using XPath filtering so that there is no unintentional
1264 validation failure due to such modifications.
- 1265 • The problem of modification by intermediaries (especially active ones) is applicable to
1266 more than just XPath processing. Digital signatures, because of canonicalization and
1267 digests, present particularly fragile examples of such relationships. If overall message
1268 processing is to remain robust, intermediaries must exercise care that the transformation
1269 algorithms used do not affect the validity of a digitally signed component.
- 1270 • Due to security concerns with namespaces, this specification strongly RECOMMENDS
1271 the use of the "Exclusive XML Canonicalization" algorithm or another canonicalization
1272 algorithm that provides equivalent or greater protection.
- 1273 • For processing efficiency it is RECOMMENDED to have the signature added and then
1274 the security token pre-pended so that a processor can read and cache the token before it
1275 is used.

1276 8.3 Signing Tokens

1277 It is often desirable to sign security tokens that are included in a message or even external to the
1278 message. The XML Signature specification provides several common ways for referencing
1279 information to be signed such as URIs, IDs, and XPath, but some token formats may not allow
1280 tokens to be referenced using URIs or IDs and XPaths may be undesirable in some situations.
1281 This specification allows different tokens to have their own unique reference mechanisms which
1282 are specified in their profile as extensions to the `<wsse:SecurityTokenReference>` element.

1283 This element provides a uniform referencing mechanism that is guaranteed to work with all token
1284 formats. Consequently, this specification defines a new reference option for XML Signature: the
1285 STR Dereference Transform.

1286
1287 This transform is specified by the URI #STR-Transform and when applied to a
1288 <wsse:SecurityTokenReference> element it means that the output is the token referenced
1289 by the <wsse:SecurityTokenReference> element not the element itself.

1290
1291 As an overview the processing model is to echo the input to the transform except when a
1292 <wsse:SecurityTokenReference> element is encountered. When one is found, the element
1293 is not echoed, but instead, it is used to locate the token(s) matching the criteria and rules defined
1294 by the <wsse:SecurityTokenReference> element and echo it (them) to the output.
1295 Consequently, the output of the transformation is the resultant sequence representing the input
1296 with any <wsse:SecurityTokenReference> elements replaced by the referenced security
1297 token(s) matched.

1298
1299 The following illustrates an example of this transformation which references a token contained
1300 within the message envelope:

```
1301 ...  
1302 <wsse:SecurityTokenReference wsu:Id="Str1">  
1303 ...  
1304 </wsse:SecurityTokenReference>  
1305 ...  
1306 ...  
1307 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
1308   <ds:SignedInfo>  
1309     ...  
1310     <ds:Reference URI="#Str1">  
1311       <ds:Transforms>  
1312         <ds:Transform  
1313           Algorithm="...#STR-Transform">  
1314             <wsse:TransformationParameters>  
1315               <ds:CanonicalizationMethod  
1316                 Algorithm="http://www.w3.org/TR/2001/REC-xml-  
1317 c14n-20010315" />  
1318             </wsse:TransformationParameters>  
1319           </ds:Transform>  
1320           <ds:DigestMethod Algorithm=  
1321             "http://www.w3.org/2000/09/xmldsig#sha1" />  
1322           <ds:DigestValue>...</ds:DigestValue>  
1323         </ds:Reference>  
1324       </ds:SignedInfo>  
1325       <ds:SignatureValue></ds:SignatureValue>  
1326     </ds:Signature>  
1327     ...
```

1328
1329 The following describes the attributes and elements listed in the example above:

1330
1331 */wsse:TransformationParameters*

1332 This element is used to wrap parameters for a transformation allows elements even from
1333 the XML Signature namespace.
1334

1335 */wsse:TransformationParameters/ds:Canonicalization*
1336 This specifies the canonicalization algorithm to apply to the selected data.
1337

1338 */wsse:TransformationParameters/{any}*
1339 This is an extensibility mechanism to allow different (extensible) parameters to be
1340 specified in the future. Unrecognized parameters SHOULD cause a fault.
1341

1342 */wsse:TransformationParameters/@{any}*
1343 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
1344 added to the element in the future. Unrecognized attributes SHOULD cause a fault.
1345

1346 The following is a detailed specification of the transformation. The algorithm is identified by the
1347 URI: #STR-Transform.
1348

1349 Transform Input:

- 1350 • The input is a node set. If the input is an octet stream, then it is automatically parsed; cf.
1351 XML Digital Signature [XMLSIG].

1352 Transform Output:

- 1353 • The output is an octet steam.

1354 Syntax:

- 1355 • The transform takes a single mandatory parameter, a
1356 `<ds:CanonicalizationMethod>` element, which is used to serialize the output node
1357 set. Note, however, that the output may not be strictly in canonical form, per the
1358 canonicalization algorithm; however, the output is canonical, in the sense that it is
1359 unambiguous. However, because of syntax requirements in the XML Signature
1360 definition, this parameter MUST be wrapped in a
1361 `<wsse:TransformationParameters>` element.
1362 •

1363 Processing Rules:

- 1364 • Let N be the input node set.
- 1365 • Let R be the set of all `<wsse:SecurityTokenReference>` elements in N.
- 1366 • For each R_i in R, let D_i be the result of dereferencing R_i .
- 1367 • If D_i cannot be determined, then the transform MUST signal a failure.
- 1368 • If D_i is an XML security token (e.g., a SAML assertion or a
1369 `<wsse:BinarySecurityToken>` element), then let R_i' be D_i . Otherwise, D_i is a raw
1370 binary security token; i.e., an octet stream. In this case, let R_i' be a node set consisting of
1371 a `<wsse:BinarySecurityToken>` element, utilizing the same namespace prefix as
1372 the `<wsse:SecurityTokenReference>` element R_i , with no `EncodingType` attribute,
1373 a `ValueType` attribute identifying the content of the security token, and text content
1374 consisting of the binary-encoded security token, with no white space.
- 1375 • Finally, employ the canonicalization method specified as a parameter to the transform to
1376 serialize N to produce the octet stream output of this transform; but, in place of any
1377 dereferenced `<wsse:SecurityTokenReference>` element R_i and its descendants,

1378 process the dereferenced node set Ri' instead. During this step, canonicalization of the
 1379 replacement node set MUST be augmented as follows:
 1380 o Note: A namespace declaration xmlns="" MUST be emitted with every apex
 1381 element that has no namespace node declaring a value for the default
 1382 namespace; cf. XML Decryption Transform.
 1383 Note: Per the processing rules above, any <wsse:SecurityTokenReference>
 1384 element is effectively replaced by the referenced <wsse:BinarySecurityToken>
 1385 element and then the <wsse:BinarySecurityToken> is canonicalized in that
 1386 context. Each <wsse:BinarySecurityToken> needs to be complete in a given
 1387 context, so any necessary namespace declarations that are not present on an ancestor
 1388 element will need to be added to the <wsse:BinarySecurityToken> element prior to
 1389 canonicalization.
 1390
 1391 Signing a <wsse:SecurityTokenReference> (STR) element provides authentication
 1392 and integrity protection of only the STR and not the referenced security token (ST). If
 1393 signing the ST is the intended behavior, the STR Dereference Transform (STRDT) may
 1394 be used which replaces the STR with the ST for digest computation, effectively protecting
 1395 the ST and not the STR. If protecting both the ST and the STR is desired, you may sign
 1396 the STR twice, once using the STRDT and once not using the STRDT.
 1397
 1398 The following table lists the full URI for each URI fragment referred to in the specification.
 1399

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

1400 8.4 Signature Validation

1401 The validation of a <ds:Signature> element inside an <wsse:Security> header block
 1402 MUST fail if:

- 1403 • the syntax of the content of the element does not conform to this specification, or
- 1404 • the validation of the signature contained in the element fails according to the core
 1405 validation of the XML Signature specification [XMLSIG], or
- 1406 • the application applying its own validation policy rejects the message for some reason
 1407 (e.g., the signature is created by an untrusted key – verifying the previous two steps only
 1408 performs cryptographic validation of the signature).

1409
 1410 If the validation of the signature element fails, applications MAY report the failure to the producer
 1411 using the fault codes defined in Section 12 Error Handling.
 1412
 1413 The signature validation shall additionally adhere to the rules defines in signature confirmation
 1414 section below, if the initiator desires signature confirmation:

1415 8.5 Signature Confirmation

1416 In the general model, the initiator uses XML Signature constructs to represent message parts of
1417 the request that were signed. The manifest of signed SOAP elements is contained in the
1418 <ds:Signature> element which in turn is placed inside the <wsse:Security> header. The
1419 <ds:Signature> element of the request contains a <ds:SignatureValue>. This element
1420 contains a base64 encoded value representing the actual digital signature. In certain situations it
1421 is desirable that initiator confirms that the message received was generated in response to a
1422 message it initiated in its unaltered form. This helps prevent certain forms of attack. This
1423 specification introduces a <wssell:SignatureConfirmation> element to address this
1424 necessity.

1425
1426 Compliant responder implementations that support signature confirmation, MUST include a
1427 <wssell:SignatureConfirmation> element inside the <wsse:Security> header of the
1428 associated response message for every <ds:Signature> element that is a direct child of the
1429 <wsse:Security> header block in the originating message. The responder MUST include the
1430 contents of the <ds:SignatureValue> element of the request signature as the value of the
1431 @Value attribute of the <wssell:SignatureConfirmation> element. The
1432 <wssell:SignatureConfirmation> element MUST be included in the message signature of
1433 the associated response message.

1434
1435 If the associated originating signature is received in encrypted form then the corresponding
1436 <wssell:SignatureConfirmation> element SHOULD be encrypted to protect the original
1437 signature and keys.

1438
1439 The schema outline for this element is as follows:

```
1440  
1441 <wssell:SignatureConfirmation wsu:Id="..." Value="..." />
```

1442
1443 */wsse11:SignatureConfirmation*

1444 This element indicates that the responder has processed the signature in the request.
1445 When this element is not present in a response the initiator SHOULD interpret that the
1446 responder is not compliant with this functionality.

1447
1448 */wsse11:SignatureConfirmation/@wsu:Id*

1449 Identifier to be used when referencing this element in the <ds:SignedInfo> reference
1450 list of the signature of the associated response message. This attribute MUST be present
1451 so that un-ambiguous references can be made to this
1452 <wssell:SignatureConfirmation> element.

1453
1454 */wsse11:SignatureConfirmation/@Value*

1455 This optional attribute contains the contents of a <ds:SignatureValue> copied from
1456 the associated request. If the request was not signed, then this attribute MUST NOT be
1457 present. If this attribute is specified with an empty value, the initiator SHOULD interpret
1458 this as incorrect behavior and process accordingly. When this attribute is not present, the
1459 initiator SHOULD interpret this to mean that the response is based on a request that was
1460 not signed.

1461 8.5.1 Response Generation Rules

1462 Conformant responders MUST include at least one `<wsse1:SignatureConfirmation>`.
1463 element in the `<wsse:Security>` header in any response(s) associated with requests. That is,
1464 the normal messaging patterns are not altered.
1465 For every response message generated, the responder MUST include a
1466 `<wsse1:SignatureConfirmation>` element for every `<ds:Signature>` element it
1467 processed from the original request message. The `Value` attribute MUST be set to the exact
1468 value of the `<ds:SignatureValue>` element of the corresponding `<ds:Signature>` element.
1469 If no `<ds:Signature>` elements are present in the original request message, the responder
1470 MUST include exactly one `<wsse1:SignatureConfirmation>` element. The `Value` attribute
1471 of the `<wsse1:SignatureConfirmation>` element MUST NOT be present. The responder
1472 MUST include all `<wsse1:SignatureConfirmation>` elements in the message signature of
1473 the response message(s). If the `<ds:Signature>` element corresponding to a
1474 `<wsse1:SignatureConfirmation>` element was encrypted in the original request message,
1475 the `<wsse1:SignatureConfirmation>` element SHOULD be encrypted for the recipient of
1476 the response message(s).
1477

1478 8.5.2 Response Processing Rules

1479 The signature validation shall additionally adhere to the following processing guidelines, if the
1480 initiator desires signature confirmation:

- 1481 • If a response message does not contain a `<wsse1:SignatureConfirmation>`
1482 element inside the `<wsse:Security>` header, the initiator SHOULD reject the response
1483 message.
- 1484 • If a response message does contain a `<wsse1:SignatureConfirmation>` element
1485 inside the `<wsse:Security>` header but `@Value` attribute is not present on
1486 `<wsse1:SignatureConfirmation>` element, and the associated request message
1487 did include a `<ds:Signature>` element, the initiator SHOULD reject the response
1488 message.
- 1489 • If a response message does contain a `<wsse1:SignatureConfirmation>` element
1490 inside the `<wsse:Security>` header and the `@Value` attribute is present on the
1491 `<wsse1:SignatureConfirmation>` element, but the associated request did not
1492 include a `<ds:Signature>` element, the initiator SHOULD reject the response
1493 message.
- 1494 • If a response message does contain a `<wsse1:SignatureConfirmation>` element
1495 inside the `<wsse:Security>` header, and the associated request message did include
1496 a `<ds:Signature>` element and the `@Value` attribute is present but does not match the
1497 stored signature value of the associated request message, the initiator SHOULD reject
1498 the response message.
- 1499 • If a response message does not contain a `<wsse1:SignatureConfirmation>`
1500 element inside the `<wsse:Security>` header corresponding to each
1501 `<ds:Signature>` element or if the `@Value` attribute present does not match the stored
1502 signature values of the associated request message, the initiator SHOULD reject the
1503 response message.

1504

8.6 Example

1505 The following sample message illustrates the use of integrity and security tokens. For this
1506 example, only the message body is signed.

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

1551

```
<?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">
        MIIIEZzCCA9CgAwIBAgIQEmtJZc0rqqrKh5i...
      </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>
          BL8jdfToEbl1/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>
```

1552

9 Encryption

1553 This specification allows encryption of any combination of body blocks, header blocks, and any of
1554 these sub-structures by either a common symmetric key shared by the producer and the recipient
1555 or a symmetric key carried in the message in an encrypted form.

1556

1557 In order to allow this flexibility, this specification leverages the XML Encryption standard. This
1558 specification describes how the two elements `<xenc:ReferenceList>` and
1559 `<xenc:EncryptedKey>` listed below and defined in XML Encryption can be used within the
1560 `<wsse:Security>` header block. When a producer or an active intermediary encrypts
1561 portion(s) of a SOAP message using XML Encryption it MUST prepend a sub-element to the
1562 `<wsse:Security>` header block. Furthermore, the encrypting party MUST either prepend the
1563 sub-element to an existing `<wsse:Security>` header block for the intended recipients or create
1564 a new `<wsse:Security>` header block and insert the sub-element. The combined process of
1565 encrypting portion(s) of a message and adding one of these sub-elements is called an encryption
1566 step hereafter. The sub-element MUST contain the information necessary for the recipient to
1567 identify the portions of the message that it is able to decrypt.

1568

1569 This specification additionally defines an element `<wsse11:EncryptedHeader>` for containing
1570 encrypted SOAP header blocks. This specification RECOMMENDS an additional mechanism that
1571 uses this element for encrypting SOAP header blocks that complies with SOAP processing
1572 guidelines while preserving the confidentiality of attributes on the SOAP header blocks.
1573 All compliant implementations MUST be able to support the XML Encryption standard [XMLENC].

1574 9.1 xenc:ReferenceList

1575 The `<xenc:ReferenceList>` element from XML Encryption [XMLENC] MAY be used to
1576 create a manifest of encrypted portion(s), which are expressed as `<xenc:EncryptedData>`
1577 elements within the envelope. An element or element content to be encrypted by this encryption
1578 step MUST be replaced by a corresponding `<xenc:EncryptedData>` according to XML
1579 Encryption. All the `<xenc:EncryptedData>` elements created by this encryption step
1580 SHOULD be listed in `<xenc:DataReference>` elements inside one or more
1581 `<xenc:ReferenceList>` element.

1582

1583 Although in XML Encryption [XMLENC], `<xenc:ReferenceList>` was originally designed to
1584 be used within an `<xenc:EncryptedKey>` element (which implies that all the referenced
1585 `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows
1586 that `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>`
1587 MAY be encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>`
1588 within individual `<xenc:EncryptedData>`.

1589

1590 A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the
1591 producer and the recipient use a shared secret key. The following illustrates the use of this sub-
1592 element:

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 45 of 76

1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613

```
<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>
```

1614

9.2 xenc:EncryptedKey

1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626

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

1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640

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

```
<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>
    </wsse:Security>
  </S11:Header>
  <S11:Body>
    ...
  </S11:Body>
</S11:Envelope>
```

```

1641         </ds:X509IssuerName>
1642 <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
1643         </ds:X509IssuerSerial>
1644         </wsse:SecurityTokenReference>
1645     </ds:KeyInfo>
1646         ...
1647     </xenc:EncryptedKey>
1648     ...
1649 </wsse:Security>
1650 </S11:Header>
1651 <S11:Body>
1652     <xenc:EncryptedData Id="bodyID">
1653         <xenc:CipherData>
1654             <xenc:CipherValue>...</xenc:CipherValue>
1655         </xenc:CipherData>
1656     </xenc:EncryptedData>
1657 </S11:Body>
1658 </S11:Envelope>

```

1659
1660 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in
1661 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that
1662 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

1663 9.3 Encrypted Header

1664 In order to be compliant with SOAP mustUnderstand processing guidelines and to prevent
1665 disclosure of information contained in attributes on a SOAP header block, this specification
1666 introduces an `<wsse11:EncryptedHeader>` element. This element contains exactly one
1667 `<xenc:EncryptedData>` element. This specification RECOMMENDS the use of
1668 `<wsse11:EncryptedHeader>` element for encrypting SOAP header blocks.

1669 9.4 Processing Rules

1670 Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the
1671 XML Encryption specification. An encrypted SOAP envelope MUST still be a valid SOAP
1672 envelope. The message creator MUST NOT encrypt the `<S11:Header>`, `<S12:Header>`,
1673 `<S11:Envelope>`, `<S12:Envelope>`, or `<S11:Body>`, `<S12:Body>` elements but MAY
1674 encrypt child elements of either the `<S11:Header>`, `<S12:Header>` and `<S11:Body>` or
1675 `<S12:Body>` elements. Multiple steps of encryption MAY be added into a single
1676 `<wsse:Security>` header block if they are targeted for the same recipient.

1677
1678 When an element or element content inside a SOAP envelope (e.g. the contents of the
1679 `<S11:Body>` or `<S12:Body>` elements) are to be encrypted, it MUST be replaced by an
1680 `<xenc:EncryptedData>`, according to XML Encryption and it SHOULD be referenced from the
1681 `<xenc:ReferenceList>` element created by this encryption step. If the target of reference is
1682 an `EncryptedHeader` as defined in section 9.3 above, see processing rules defined in section
1683 9.5.3 Encryption using `EncryptedHeader` and section 9.5.4 Decryption of `EncryptedHeader`
1684 below.

1685 9.4.1 Encryption

1686 The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1687 this specification are listed below (note that use of `<xenc:ReferenceList>` is
1688 RECOMMENDED. Additionally, if the target of encryption is a SOAP header, processing rules
1689 defined in section 9.5.3 SHOULD be used).

- 1690 • Create a new SOAP envelope.
- 1691 • Create a `<wsse:Security>` header
- 1692 • When an `<xenc:EncryptedKey>` is used, create a `<xenc:EncryptedKey>` sub-
1693 element of the `<wsse:Security>` element. This `<xenc:EncryptedKey>` sub-
1694 element SHOULD contain an `<xenc:ReferenceList>` sub-element, containing a
1695 `<xenc:DataReference>` to each `<xenc:EncryptedData>` element that was
1696 encrypted using that key.
- 1697 • Locate data items to be encrypted, i.e., XML elements, element contents within the target
1698 SOAP envelope.
- 1699 • Encrypt the data items as follows: For each XML element or element content within the
1700 target SOAP envelope, encrypt it according to the processing rules of the XML
1701 Encryption specification [XMLENC]. Each selected original element or element content
1702 MUST be removed and replaced by the resulting `<xenc:EncryptedData>` element.
- 1703 • The optional `<ds:KeyInfo>` element in the `<xenc:EncryptedData>` element MAY
1704 reference another `<ds:KeyInfo>` element. Note that if the encryption is based on an
1705 attached security token, then a `<wsse:SecurityTokenReference>` element SHOULD
1706 be added to the `<ds:KeyInfo>` element to facilitate locating it.
- 1707 • Create an `<xenc:DataReference>` element referencing the generated
1708 `<xenc:EncryptedData>` elements. Add the created `<xenc:DataReference>`
1709 element to the `<xenc:ReferenceList>`.
- 1710 • Copy all non-encrypted data.

1711 9.4.2 Decryption

1712 On receiving a SOAP envelope containing encryption header elements, for each encryption
1713 header element the following general steps should be processed (this section is non-normative.
1714 Additionally, if the target of reference is an `EncryptedHeader`, processing rules as defined in
1715 section 9.5.4 below SHOULD be used):

- 1716
- 1717 1. Identify any decryption keys that are in the recipient's possession, then identifying any
1718 message elements that it is able to decrypt.
- 1719 2. Locate the `<xenc:EncryptedData>` items to be decrypted (possibly using the
1720 `<xenc:ReferenceList>`).
- 1721 3. Decrypt them as follows:
 - 1722 a. For each element in the target SOAP envelope, decrypt it according to the
1723 processing rules of the XML Encryption specification and the processing rules
1724 listed above.
 - 1725 b. If the decryption fails for some reason, applications MAY report the failure to the
1726 producer using the fault code defined in Section 12 Error Handling of this
1727 specification.

1728 c. It is possible for overlapping portions of the SOAP message to be encrypted in
1729 such a way that they are intended to be decrypted by SOAP nodes acting in
1730 different Roles. In this case, the <xenc:ReferenceList> or
1731 <xenc:EncryptedKey> elements identifying these encryption operations will
1732 necessarily appear in different <wsse:Security> headers. Since SOAP does
1733 not provide any means of specifying the order in which different Roles will
1734 process their respective headers, this order is not specified by this specification
1735 and can only be determined by a prior agreement.

1736 9.4.3 Encryption with EncryptedHeader

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

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 In addition, <wsse11:EncryptedHeader> header blocks can be super-encrypted and replaced
1755 by other <wsse11:EncryptedHeader> header blocks (for wrapping/tunneling scenarios). Any
1756 <wsse:Security> header that encrypts a header block targeted to a particular actor SHOULD
1757 be targeted to that same actor, unless it is a security header.
1758
1759

1760 9.4.4 Processing an EncryptedHeader

1761 The processing model for <wsse11:EncryptedHeader> header blocks is as follows:

- 1762 1. Resolve references to encrypted data specified in the <wsse:Security> header block
1763 targeted at this node. For each reference, perform the following steps.
- 1764 2. If the referenced element does not have a qualified name of
1765 <wsse11:EncryptedHeader> then process as per section 9.5.2 Decryption and stop
1766 the processing steps here.
- 1767 3. Otherwise, extract the <xenc:EncryptedData> element from the
1768 <wsse11:EncryptedHeader> element.

1769 4. Decrypt the contents of the `<xenc:EncryptedData>` element as per section 9.5.2
1770 Decryption and replace the `<wsse11:EncryptedHeader>` element with the decrypted
1771 contents.

1772 5. Process the decrypted header block as per SOAP processing guidelines.

1773

1774 Alternatively, a processor may perform a pre-pass over the encryption references in the
1775 `<wsse:Security>` header:

1776 1. Resolve references to encrypted data specified in the `<wsse:Security>` header block
1777 targeted at this node. For each reference, perform the following steps.

1778 2. If a referenced element has a qualified name of `<wsse11:EncryptedHeader>` then
1779 replace the `<wsse11:EncryptedHeader>` element with the contained
1780 `<xenc:EncryptedData>` element and if present copy the value of the `wsu:Id` attribute
1781 from the `<wsse11:EncryptedHeader>` element to the `<xenc:EncryptedData>`
1782 element.

1783 3. Process the `<wsse:Security>` header block as normal.

1784

1785 It should be noted that the results of decrypting a `<wsse11:EncryptedHeader>` header block
1786 could be another `<wsse11:EncryptedHeader>` header block. In addition, the result MAY be
1787 targeted at a different role than the role processing the `<wsse11:EncryptedHeader>` header
1788 block.

1789 **9.4.5 Processing the mustUnderstand attribute on EncryptedHeader**

1790 If the `S11:mustUnderstand` or `S12:mustUnderstand` attribute is specified on the
1791 `<wsse11:EncryptedHeader>` header block, and is true, then the following steps define what it
1792 means to "understand" the `<wsse11:EncryptedHeader>` header block:

1793 1. The processor MUST be aware of this element and know how to decrypt and convert into
1794 the original header block. This DOES NOT REQUIRE that the process know that it has
1795 the correct keys or support the indicated algorithms.

1796 2. The processor MUST, after decrypting the encrypted header block, process the
1797 decrypted header block according to the SOAP processing guidelines. The receiver
1798 MUST raise a fault if any content required to adequately process the header block
1799 remains encrypted or if the decrypted SOAP header is not understood and the value of
1800 the `S12:mustUnderstand` or `S11:mustUnderstand` attribute on the decrypted
1801 header block is true. Note that in order to comply with SOAP processing rules in this
1802 case, the processor must roll back any persistent effects of processing the security
1803 header, such as storing a received token.

1804

1805

10 Security Timestamps

1806 It is often important for the recipient to be able to determine the *freshness* of security semantics.
1807 In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.
1808 This specification does not provide a mechanism for synchronizing time. The assumption is that
1809 time is trusted or additional mechanisms, not described here, are employed to prevent replay.
1810 This specification defines and illustrates time references in terms of the `xsd:dateTime` type
1811 defined in XML Schema. It is RECOMMENDED that all time references use this type. All
1812 references MUST be in UTC time. Implementations MUST NOT generate time instants that
1813 specify leap seconds. If, however, other time types are used, then the `ValueType` attribute
1814 (described below) MUST be specified to indicate the data type of the time format. Requestors and
1815 receivers SHOULD NOT rely on other applications supporting time resolution finer than
1816 milliseconds.

1817

1818 The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and
1819 expiration times of the security semantics in a message.

1820

1821 All times MUST be in UTC format as specified by the XML Schema type (`dateTime`). It should be
1822 noted that times support time precision as defined in the XML Schema specification.

1823 The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and
1824 may only be present at most once per header (that is, per SOAP actor/role).

1825

1826 The ordering within the element is as illustrated below. The ordering of elements in the
1827 `<wsu:Timestamp>` element is fixed and MUST be preserved by intermediaries.

1828 The schema outline for the `<wsu:Timestamp>` element is as follows:

1829

```
1830 <wsu:Timestamp wsu:Id="...">  
1831   <wsu:Created ValueType="...">...</wsu:Created>  
1832   <wsu:Expires ValueType="...">...</wsu:Expires>  
1833   ...  
1834 </wsu:Timestamp>
```

1835

1836 The following describes the attributes and elements listed in the schema above:

1837

1838 */wsu:Timestamp*

1839 This is the element for indicating security semantics timestamps.

1840

1841 */wsu:Timestamp/wsui:Created*

1842 This represents the creation time of the security semantics. This element is optional, but
1843 can only be specified once in a `<wsu:Timestamp>` element. Within the SOAP
1844 processing model, creation is the instant that the infoset is serialized for transmission.

1845 The creation time of the message SHOULD NOT differ substantially from its transmission
1846 time. The difference in time should be minimized.

1847

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

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

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

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

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

1880
1881 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```
1882  
1883 <S11:Envelope xmlns:S11="..." xmlns:wssse="..." xmlns:wsu="...">  
1884 <S11:Header>  
1885 <wsse:Security>  
1886 <wsu:Timestamp wsu:Id="timestamp">  
1887 <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>  
1888 <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>  
1889 </wsu:Timestamp>  
1890 ...  
1891 </wsse:Security>  
1892 ...  
1893 </S11:Header>  
1894 <S11:Body>  
1895 ...  
1896 </S11:Body>
```

1897

</S11:Envelope>

1898

11 Extended Example

1899 The following sample message illustrates the use of security tokens, signatures, and encryption.
1900 For this example, the timestamp and the message body are signed prior to encryption. The
1901 decryption transformation is not needed as the signing/encryption order is specified within the
1902 <wsse:Security> header.

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

1946

```
(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)           </wsse:SecurityTokenReference>
(026)         </ds:KeyInfo>
(027)         <xenc:CipherData>
(028)           <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(029)         </xenc:CipherValue>
(030)       </xenc:EncryptedKey>
(031)       <xenc:ReferenceList>
(032)         <xenc:DataReference URI="#enc1"/>
(033)       </xenc:ReferenceList>
(034)     </wsse:Security>
(035)   </S11:Header>
(036) </S11:Envelope>
```

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 54 of 76

```

1947 (033)         <ds:Transform
1948         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1949         </ds:Transforms>
1950 (034)         <ds:DigestMethod
1951         Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1952         <ds:DigestValue>LyLsF094hPi4wPU...
1953         </ds:DigestValue>
1954         </ds:Reference>
1955         <ds:Reference URI="#body">
1956         <ds:Transforms>
1957         <ds:Transform
1958         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1959         </ds:Transforms>
1960         <ds:DigestMethod
1961         Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1962         <ds:DigestValue>LyLsF094hPi4wPU...
1963         </ds:DigestValue>
1964         </ds:Reference>
1965         </ds:SignedInfo>
1966         <ds:SignatureValue>
1967         Hp1ZkmFZ/2kQLXDJbchm5gK...
1968         </ds:SignatureValue>
1969         <ds:KeyInfo>
1970         <wsse:SecurityTokenReference>
1971         <wsse:Reference URI="#X509Token" />
1972         </wsse:SecurityTokenReference>
1973         </ds:KeyInfo>
1974         </ds:Signature>
1975         </wsse:Security>
1976 </S11:Header>
1977 <S11:Body wsu:Id="body">
1978 <xenc:EncryptedData
1979     Type="http://www.w3.org/2001/04/xmlenc#Element"
1980     wsu:Id="enc1">
1981 (061) <xenc:EncryptionMethod
1982     Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
1983 cbc" />
1984 (062) <xenc:CipherData>
1985 (063) <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1986 (064) </xenc:CipherValue>
1987 (065) </xenc:CipherData>
1988 (066) </xenc:EncryptedData>
1989 (067) </S11:Body>
1990 (068) </S11:Envelope>

```

1991
1992
1993
1994
1995
1996
1997
1998
1999

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.

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 55 of 76

2000
2001 Lines (010)-(012) specify a security token that is associated with the message. In this case, it
2002 specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64
2003 encoding of the certificate.
2004
2005 Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a
2006 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
2007 encrypt the key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the
2008 symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines
2009 (023)-(025) identify the encryption block in the message that uses this symmetric key. In this
2010 case it is only used to encrypt the body (Id="enc1").
2011
2012 Lines (027)-(056) specify the digital signature. In this example, the signature is based on the
2013 X.509 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039)
2014 references the message body.
2015
2016 Lines (048)-(050) indicate the actual signature value – specified in Line (043).
2017
2018 Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the X.509
2019 certificate included in the message. Line (053) provides a URI link to the Lines (010)-(012).
2020 The body of the message is represented by Lines (059)-(067).
2021
2022 Lines (060)-(066) represent the encrypted metadata and form of the body using XML Encryption.
2023 Line (060) indicates that the "element value" is being replaced and identifies this encryption. Line
2024 (061) specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the
2025 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
2026 key as the key references this encryption – Line (024).
2027

2028

12 Error Handling

2029

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

2030

For example:

2031

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

2032

- Invalid or unauthenticated or unauthenticatable security token

2033

- Invalid signature

2034

- Decryption failure

2035

- Referenced security token is unavailable

2036

- Unsupported namespace

2037

2038

If a service does not perform its normal operation because of the contents of the Security header, then that MAY be reported using SOAP's Fault Mechanism. This specification does not mandate that faults be returned as this could be used as part of a denial of service or cryptographic attack. We combine signature and encryption failures to mitigate certain types of attacks.

2039

2040

2041

2042

2043

If a failure is returned to a producer then the failure MUST be reported using the SOAP Fault mechanism. The following tables outline the predefined security fault codes. The "unsupported" classes of errors are as follows. Note that the reason text provided below is RECOMMENDED, but alternative text MAY be provided if more descriptive or preferred by the implementation. The tables below are defined in terms of SOAP 1.1. For SOAP 1.2, the Fault/Code/Value is *env:Sender* (as defined in SOAP 1.2) and the Fault/Code/Subcode/Value is the *faultcode* below and the Fault/Reason/Text is the *faultstring* below.

2044

2045

2046

2047

2048

2049

2050

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

2051

2052

The "failure" class of errors are:

2053

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

2054

13 Security Considerations

2055

2056

2057

2058

2059

2060

2061

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.

2062

13.1 General Considerations

2063

2064

2065

2066

2067

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*.

2068

2069

2070

2071

2072

2073

2074

2075

2076

2077

2078

2079

2080

2081

2082

- 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)

2083

2084

2085

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.

2086

13.2 Additional Considerations

2087

13.2.1 Replay

2088

2089

2090

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

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 59 of 76

2091 messages are exchanged via an open network. These can be part of the message or of the
2092 headers defined from other SOAP extensions. Four typical approaches are: Timestamp,
2093 Sequence Number, Expirations and Message Correlation. Signed timestamps MAY be used to
2094 keep track of messages (possibly by caching the most recent timestamp from a specific service)
2095 and detect replays of previous messages. It is RECOMMENDED that timestamps be cached for
2096 a given period of time, as a guideline, a value of five minutes can be used as a minimum to detect
2097 replays, and that timestamps older than that given period of time set be rejected in interactive
2098 scenarios.

2099 **13.2.2 Combining Security Mechanisms**

2100 This specification defines the use of XML Signature and XML Encryption in SOAP headers. As
2101 one of the building blocks for securing SOAP messages, it is intended to be used in conjunction
2102 with other security techniques. Digital signatures need to be understood in the context of other
2103 security mechanisms and possible threats to an entity.

2104
2105 Implementers should also be aware of all the security implications resulting from the use of digital
2106 signatures in general and XML Signature in particular. When building trust into an application
2107 based on a digital signature there are other technologies, such as certificate evaluation, that must
2108 be incorporated, but these are outside the scope of this document.

2109
2110 As described in XML Encryption, the combination of signing and encryption over a common data
2111 item may introduce some cryptographic vulnerability. For example, encrypting digitally signed
2112 data, while leaving the digital signature in the clear, may allow plain text guessing attacks.

2113 **13.2.3 Challenges**

2114 When digital signatures are used for verifying the claims pertaining to the sending entity, the
2115 producer must demonstrate knowledge of the confirmation key. One way to achieve this is to use
2116 a challenge-response type of protocol. Such a protocol is outside the scope of this document.
2117 To this end, the developers can attach timestamps, expirations, and sequences to messages.

2118 **13.2.4 Protecting Security Tokens and Keys**

2119 Implementers should be aware of the possibility of a token substitution attack. In any situation
2120 where a digital signature is verified by reference to a token provided in the message, which
2121 specifies the key, it may be possible for an unscrupulous producer to later claim that a different
2122 token, containing the same key, but different information was intended.

2123 An example of this would be a user who had multiple X.509 certificates issued relating to the
2124 same key pair but with different attributes, constraints or reliance limits. Note that the signature of
2125 the token by its issuing authority does not prevent this attack. Nor can an authority effectively
2126 prevent a different authority from issuing a token over the same key if the user can prove
2127 possession of the secret.

2128
2129 The most straightforward counter to this attack is to insist that the token (or its unique identifying
2130 data) be included under the signature of the producer. If the nature of the application is such that
2131 the contents of the token are irrelevant, assuming it has been issued by a trusted authority, this

2132 attack may be ignored. However because application semantics may change over time, best
2133 practice is to prevent this attack.

2134
2135 Requestors should use digital signatures to sign security tokens that do not include signatures (or
2136 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
2137 RECOMMENDED that all relevant and immutable message content be signed by the producer.
2138 Receivers SHOULD only consider those portions of the document that are covered by the
2139 producer's signature as being subject to the security tokens in the message. Security tokens
2140 appearing in <wsse:Security> header elements SHOULD be signed by their issuing authority
2141 so that message receivers can have confidence that the security tokens have not been forged or
2142 altered since their issuance. It is strongly RECOMMENDED that a message producer sign any
2143 <wsse:SecurityToken> elements that it is confirming and that are not signed by their issuing
2144 authority.
2145 When a requester provides, within the request, a Public Key to be used to encrypt the response,
2146 it is possible that an attacker in the middle may substitute a different Public Key, thus allowing the
2147 attacker to read the response. The best way to prevent this attack is to bind the encryption key in
2148 some way to the request. One simple way of doing this is to use the same key pair to sign the
2149 request as to encrypt the response. However, if policy requires the use of distinct key pairs for
2150 signing and encryption, then the Public Key provided in the request should be included under the
2151 signature of the request.

2152 **13.2.5 Protecting Timestamps and Ids**

2153 In order to *trust* wsu:Id attributes and <wsu:Timestamp> elements, they SHOULD be signed
2154 using the mechanisms outlined in this specification. This allows readers of the IDs and
2155 timestamps information to be certain that the IDs and timestamps haven't been forged or altered
2156 in any way. It is strongly RECOMMENDED that IDs and timestamp elements be signed.
2157

2158 **13.2.6 Protecting against removal and modification of XML Elements**

2159 XML Signatures using Shorthand XPointer References (AKA IDREF) protect against the removal
2160 and modification of XML elements; but do not protect the location of the element within the XML
2161 Document.

2162
2163 Whether or not this is a security vulnerability depends on whether the location of the signed data
2164 within its surrounding context has any semantic import. This consideration applies to data carried
2165 in the SOAP Body or the Header.

2166
2167 Of particular concern is the ability to relocate signed data into a SOAP Header block which is
2168 unknown to the receiver and marked mustUnderstand="false". This could have the effect of
2169 causing the receiver to ignore signed data which the sender expected would either be processed
2170 or result in the generation of a MustUnderstand fault.

2171
2172 A similar exploit would involve relocating signed data into a SOAP Header block targeted to a
2173 S11:actor or S12:role other than that which the sender intended, and which the receiver will not
2174 process.
2175

2176 While these attacks could apply to any portion of the message, their effects are most pernicious
2177 with SOAP header elements which may not always be present, but must be processed whenever
2178 they appear.

2179

2180 In the general case of XML Documents and Signatures, this issue may be resolved by signing the
2181 entire XML Document and/or strict XML Schema specification and enforcement. However,
2182 because elements of the SOAP message, particularly header elements, may be legitimately
2183 modified by SOAP intermediaries, this approach is usually not appropriate. It is RECOMMENDED
2184 that applications signing any part of the SOAP body sign the entire body.

2185

2186 Alternatives countermeasures include (but are not limited to):

- 2187 • References using XPath transforms with Absolute Path expressions with checks
2188 performed by the receiver that the URI and Absolute Path XPath expression evaluate to
2189 the digested nodeset.
- 2190 • A Reference using an XPath transform to include any significant location-dependent
2191 elements and exclude any elements that might legitimately be removed, added, or altered
2192 by intermediaries,
- 2193 • Using only References to elements with location-independent semantics,
- 2194 • Strict policy specification and enforcement regarding which message parts are to be
2195 signed. For example:
 - 2196 ○ Requiring that the entire SOAP Body and all children of SOAP Header be signed,
 - 2197 ○ Requiring that SOAP header elements which are marked
2198 `MustUnderstand="false"` and have signed descendants **MUST** include the
2199 `MustUnderstand` attribute under the signature.

2200

2201 **13.2.7 Detecting Duplicate Identifiers**

2202 The `<wsse:Security>` processing **SHOULD** check for duplicate values from among the set of
2203 ID attributes that it is aware of. The `wsse:Security` processing **MUST** generate a fault if a
2204 duplicate ID value is detected.

2205

2206 This section is non-normative.

2207

14 Interoperability Notes

2208

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.

2209

2210

2211

2212

2213

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

2214

2215

- **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.

2216

2217

2218

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

2219

2220

2221

- **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.

2222

2223

2224

2225

2226

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

2227

2228

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

2229

2230

- **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.

2231

2232

2233

2234

This section is non-normative.

2235

15 Privacy Considerations

2236

In the context of this specification, we are only concerned with potential privacy violation by the security elements defined here. Privacy of the content of the payload message is out of scope.

2237

2238

Producers or sending applications should be aware that claims, as collected in security tokens, are typically personal information, and should thus only be sent according to the producer's

2239

2240

privacy policies. Future standards may allow privacy obligations or restrictions to be added to this data. Unless such standards are used, the producer must ensure by out-of-band means that the

2241

2242

recipient is bound to adhering to all restrictions associated with the data, and the recipient must

2243

2244

similarly ensure by out-of-band means that it has the necessary consent for its intended

2245

2246

If claim data are visible to intermediaries, then the policies must also allow the release to these intermediaries. As most personal information cannot be released to arbitrary parties, this will

2247

2248

typically require that the actors are referenced in an identifiable way; such identifiable references are also typically needed to obtain appropriate encryption keys for the intermediaries.

2249

2250

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

2251

2252

2253

Intermediaries may also gain traffic information from a SOAP message exchange, e.g., who

2254

2255

communicates with whom at what time. Producers that use intermediaries should verify that

2256

2257

This section is non-normative.

2258 16 References

- 2259 **[GLOSS]** Informational RFC 2828, "Internet Security Glossary," May 2000.
- 2260 **[KERBEROS]** J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993, <http://www.ietf.org/rfc/rfc1510.txt> .
- 2261
- 2262 **[KEYWORDS]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997.
- 2263
- 2264 **[SHA-1]** FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce / National Institute of Standards and Technology. <http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>
- 2265
- 2266
- 2267 **[SOAP11]** W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.
- 2268 **[SOAP12]** W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework", 23 June 2003.
- 2269
- 2270 **[SOAPSEC]** W3C Note, "SOAP Security Extensions: Digital Signature," 06 February 2001.
- 2271
- 2272 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 3986, MIT/LCS, Day Software, Adobe Systems, January 2005.
- 2273
- 2274
- 2275 **[XPath]** W3C Recommendation, "XML Path Language", 16 November 1999
- 2276
- 2277 The following are non-normative references included for background and related material:
- 2278 **[WS-SECURITY]** "Web Services Security Language", IBM, Microsoft, VeriSign, April 2002.
- 2279 "WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002.
- 2280 "WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002.
- 2281 **[XMLC14N]** W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001.
- 2282 **[EXCC14N]** W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8 July 2002.
- 2283
- 2284 **[XMLENC]** W3C Working Draft, "XML Encryption Syntax and Processing," 04 March 2002.
- 2285
- 2286 W3C Recommendation, "Decryption Transform for XML Signature", 10 December 2002.
- 2287 **[XML-ns]** W3C Recommendation, "Namespaces in XML," 14 January 1999.
- 2288 **[XMLSCHEMA]** W3C Recommendation, "XML Schema Part 1: Structures," 2 May 2001.
- 2289 W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.
- 2290 **[XMLSIG]** D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-Signature Syntax and Processing*, W3C Recommendation, 12 February 2002.
- 2291
- 2292

WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
Page 65 of 76

2293	[X509]	S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified Certificates Profile,"
2294		
2295		http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-1
2296		
2297	[WSS-SAML]	OASIS Working Draft 06, "Web Services Security SAML Token Profile",
2298		21 February 2003
2299	[WSS-XrML]	OASIS Working Draft 03, "Web Services Security XrML Token Profile",
2300		30 January 2003
2301	[WSS-X509]	OASIS, "Web Services Security X.509 Certificate Token Profile", 19
2302		January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0
2303		
2304	[WSSKERBEROS]	OASIS Working Draft 03, "Web Services Security Kerberos Profile", 30
2305		January 2003
2306	[WSSUSERNAME]	OASIS, "Web Services Security UsernameToken Profile" 19 January
2307		2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
2308		
2309	[WSS-XCBF]	OASIS Working Draft 1.1, "Web Services Security XCBF Token Profile",
2310		30 March 2003
2311	[XMLID]	W3C Recommendation, "xml:id Version 1.0", 9 September 2005.
2312	[XPOINTER]	"XML Pointer Language (XPointer) Version 1.0, Candidate
2313		Recommendation", DeRose, Maler, Daniel, 11 September 2001.

2314

Appendix A: Acknowledgements

2315

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

WSS: SOAP Message Security (WS-Security 2004)
 Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
 Page 67 of 76

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

2316 **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

WSS: SOAP Message Security (WS-Security 2004)
 Copyright © OASIS Open 2002-2006. All Rights Reserved.

1 February 2006
 Page 68 of 76

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

2317

2318

Appendix B: Revision History

2319

2320

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

2321

Appendix C: Utility Elements and Attributes

2322
2323
2324
2325
2326

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

2327

16.1 Identification Attribute

2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339

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

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

2340
2341
2342
2343
2344
2345

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

This section is non-normative.

2346

16.2 Timestamp Elements

2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357

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

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

The following table provides an overview of these elements:

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

<wsu:Expires>	This element is used to indicate the expiration time associated with the enclosing context.
---------------	---

2358
2359
2360
2361
2362

A detailed description can be found in Section 10.

This section is non-normative.

2363 16.3 General Schema Types

2364 The schema for the utility aspects of this specification also defines some general purpose
2365 schema elements. While these elements are defined in this schema for use with this
2366 specification, they are general purpose definitions that may be used by other specifications as
2367 well.

2368
2369
2370

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.

2371
2372
2373

This section is non-normative.

2374

Appendix D: SecurityTokenReference Model

2375 This appendix provides a non-normative overview of the usage and processing models for the
2376 <wsse:SecurityTokenReference> element.

2377

2378 There are several motivations for introducing the <wsse:SecurityTokenReference>
2379 element:

2380

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

2381

2382

2383

2384

2385

2386

2387

2388

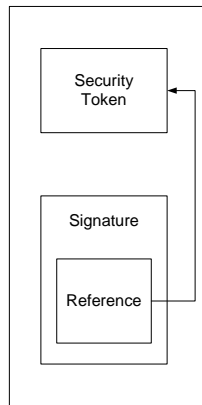
2389

2390

2391 The following use cases drive the above motivations:

2392

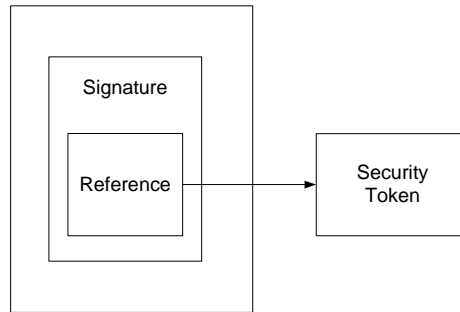
2393 **Local Reference** – A security token, that is included in the message in the <wsse:Security>
2394 header, is associated with an XML Signature. The figure below illustrates this:



2395

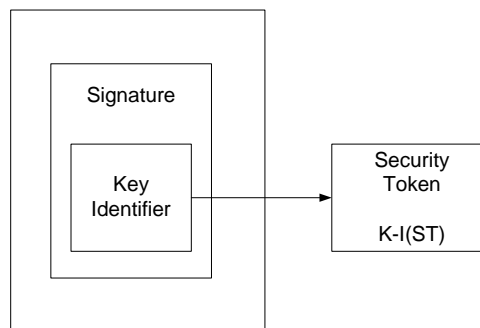
2396
2397
2398
2399

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:



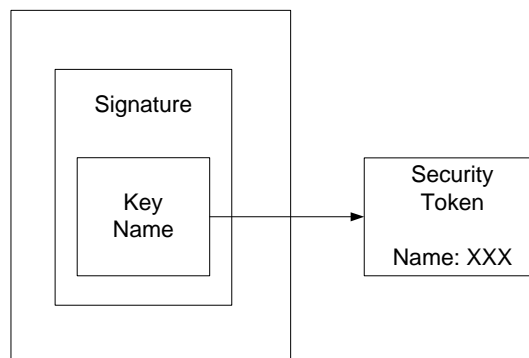
2400
2401
2402
2403

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:



2404
2405
2406
2407

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:

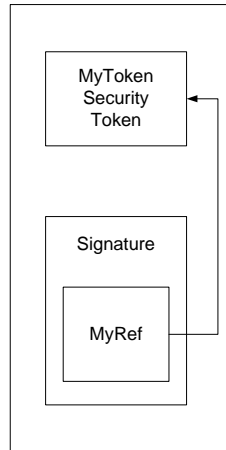


2408
2409
2410

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 2002-2006. All Rights Reserved.

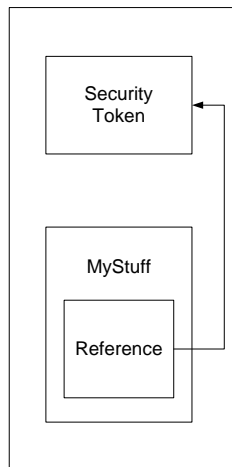
1 February 2006
Page 74 of 76



2411 described above). The figure below illustrates this:

2412

2413 **Non-Signature References** – A message may contain XML that does not represent an XML



2414 signature, but may reference a security token (which may or may not be included in the
2415 message). The figure below illustrates this:

2416

2417

2418 All conformant implementations must be able to process the
2419 `<wsse:SecurityTokenReference>` element. However, they are not required to support all of
2420 the different types of references.

2421

2422 The reference may include a `wsse11:TokenType` attribute which provides a "hint" for the type of
2423 desired token.

2424

2425 If multiple sub-elements are specified, together they describe the reference for the token.

2426 There are several challenges that implementations face when trying to interoperate:

2427 **ID References** – The underlying XML referencing mechanism using the XML base type of ID
2428 provides a simple straightforward XML element reference. However, because this is an XML
2429 type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
2430 requires the recipient to *understand* the schema. This may be an expensive task and in the
2431 general case impossible as there is no way to know the "schema location" for a specific
2432 namespace URI.

2433

2434 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
2435 references are, by definition, unique by XML. However, other mechanisms such as "principal
2436 name" are not required to be unique and therefore such references may be unique.
2437 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
2438 information about the "key" used in the signature. For token references within signatures, it is
2439 recommended that the `<wsse:SecurityTokenReference>` be placed within the
2440 `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
2441 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
2442 Message Security or its profiles are preferred over the mechanisms in XML Signature.
2443 The following provides additional details on the specific reference mechanisms defined in WSS:
2444 SOAP Message Security:

2445
2446 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
2447 the security token. If only the fragment is specified, then it references the security token within
2448 the document whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to
2449 a [potentially external] security token identified using a URI. There are no implied semantics
2450 around the processing of the URI.

2451
2452 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
2453 by specifying a known value (identifier) for the token, which is determined by applying a special
2454 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
2455 specific security token but requires a profile or token-specific function to be specified. The
2456 `ValueType` attribute defines the type of key identifier and, consequently, identifies the type of
2457 token referenced. The `EncodingType` attribute specifies how the unique value (identifier) is
2458 encoded. For example, a hash value may be encoded using base 64 encoding.

2459
2460 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
2461 specific value that is used to *match* an identity assertion within the security token. This is a
2462 subset match and may result in multiple security tokens that match the specified name. While
2463 XML Signature doesn't imply formatting semantics, WSS: SOAP Message Security recommends
2464 that X.509 names be specified.

2465
2466 It is expected that, where appropriate, profiles define if and how the reference mechanisms map
2467 to the specific token profile. Specifically, the profile should answer the following questions:

- 2468
2469
- What types of references can be used?
 - How "Key Name" references map (if at all)?
 - How "Key Identifier" references map (if at all)?
 - Are there any additional profile or format-specific references?
- 2470
2471
2472

2473
2474 This section is non-normative.