



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

Draft - 11 October 2005

Deleted: OASIS Public Review

Deleted: 28

Deleted: J

Deleted: une

OASIS identifier:

{product-productVersion-artifactType-stage-descriptiveName-revision.form (Word) (PDF)
(HTML)}

Location:

<http://docs.oasis-open.org/wss/2005/xx/wss-v1.1-spec-draft-SOAPMessageSecurity-01>

Deleted: http://docs.oasis-open.org/wss/2005/xx/wss-v1.1-spec-pr-

Technical Committee:

Web Service Security (WSS)

Formatted: Default Paragraph Font

Formatted: Font: Not Bold

Chairs:

Kelvin Lawrence, IBM
Chris Kaler, Microsoft

Formatted: Font: Not Bold

Editors:

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

Formatted: Font: Not Bold

Formatted: Font: Bold

Abstract:

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

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

Deleted: 28

Deleted: June

32
33
34
35
36

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

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

Status:

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

Deleted: 28

Deleted: June

50 Notices

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

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

65 Copyright © OASIS Open 2002-2005. *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 must not be modified in any way, such as by removing the
 72 copyright 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

Deleted: does

77 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
 78 successors or assignees.
 79

Deleted: ns

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 **This section is non-normative.**

Deleted: 28

Deleted: June

Table of Contents

88	<u>1 Introduction.....</u>	<u>7</u>
89	<u>1.1 Goals and Requirements</u>	<u>7</u>
90	<u>1.1.1 Requirements</u>	<u>8</u>
91	<u>1.1.2 Non-Goals</u>	<u>8</u>
92	<u>2 Notations and Terminology.....</u>	<u>9</u>
93	<u>2.1 Notational Conventions</u>	<u>9</u>
94	<u>2.2 Namespaces</u>	<u>9</u>
95	<u>2.3 Acronyms and Abbreviations</u>	<u>10</u>
96	<u>2.4 Terminology</u>	<u>10</u>
97	<u>2.5 Note on Examples</u>	<u>12</u>
98	<u>3 Message Protection Mechanisms.....</u>	<u>13</u>
99	<u>3.1 Message Security Model.....</u>	<u>13</u>
100	<u>3.2 Message Protection</u>	<u>13</u>
101	<u>3.3 Invalid or Missing Claims</u>	<u>14</u>
102	<u>3.4 Example.....</u>	<u>14</u>
103	<u>4 ID References</u>	<u>17</u>
104	<u>4.1 Id Attribute</u>	<u>17</u>
105	<u>4.2 Id Schema</u>	<u>17</u>
106	<u>5 Security Header.....</u>	<u>19</u>
107	<u>6 Security Tokens.....</u>	<u>22</u>
108	<u>6.1 Attaching Security Tokens</u>	<u>22</u>
109	<u>6.1.1 Processing Rules.....</u>	<u>22</u>
110	<u>6.1.2 Subject Confirmation</u>	<u>22</u>
111	<u>6.2 User Name Token.....</u>	<u>22</u>
112	<u>6.2.1 Usernames</u>	<u>22</u>
113	<u>6.3 Binary Security Tokens</u>	<u>23</u>
114	<u>6.3.1 Attaching Security Tokens</u>	<u>23</u>
115	<u>6.3.2 Encoding Binary Security Tokens</u>	<u>23</u>
116	<u>6.4 XML Tokens.....</u>	<u>25</u>
117	<u>6.5 EncryptedData Token</u>	<u>25</u>
118	<u>6.6 Identifying and Referencing Security Tokens</u>	<u>25</u>
119	<u>7 Token References</u>	<u>26</u>
120	<u>7.1 SecurityTokenReference Element</u>	<u>26</u>
121	<u>7.2 Direct References</u>	<u>28</u>
122	<u>7.3 Key Identifiers</u>	<u>29</u>

Deleted: 28

Deleted: June

123	7.4 Embedded References	31
124	7.5 ds:KeyInfo.....	32
125	7.6 Key Names	32
126	7.7 Encrypted Key reference	32
127	8 Signatures	34
128	8.1 Algorithms.....	34
129	8.2 Signing Messages	37
130	8.3 Signing Tokens	37
131	8.4 Signature Validation.....	40
132	8.5 Signature Confirmation	40
133	8.5.1 Response Generation Rules.....	41
134	8.5.2 Response Processing Rules.....	42
135	8.6 Example.....	42
136	9 Encryption	44
137	9.1 xenc:ReferenceList	44
138	9.2 xenc:EncryptedKey	45
139	9.3 Encrypted Header.....	46
140	9.4 Processing Rules.....	46
141	9.4.1 Encryption	46
142	9.4.2 Decryption	47
143	9.4.3 Encryption with EncryptedHeader.....	48
144	9.4.4 Processing an EncryptedHeader	48
145	9.4.5 Processing the mustUnderstand attribute on EncryptedHeader	49
146	10 Security Timestamps	50
147	11 Extended Example	52
148	12 Error Handling	55
149	13 Security Considerations.....	57
150	13.1 General Considerations	57
151	13.2 Additional Considerations	57
152	13.2.1 Replay	57
153	13.2.2 Combining Security Mechanisms.....	58
154	13.2.3 Challenges.....	58
155	13.2.4 Protecting Security Tokens and Keys	58
156	13.2.5 Protecting Timestamps and Ids	59
157	13.2.6 Protecting against removal and modification of XML Elements	59
158	13.2.7 Detecting Duplicate Identifiers	60
159	14 Interoperability Notes.....	61
160	15 Privacy Considerations.....	62
161	16 References	63

Deleted: 28
 Deleted: June

162	Appendix A: Acknowledgements.....	65
163	Appendix B: Revision History.....	68
164	Appendix C: Utility Elements and Attributes.....	69
165	16.1 Identification Attribute.....	69
166	16.2 Timestamp Elements.....	69
167	16.3 General Schema Types.....	70
168	Appendix D: SecurityTokenReference.....	71
169	▼-----	

Deleted: 1	Introduction	71
1.1	Goals and Requirements	71
1.1.1	Requirements	81
1.1.2	Non-Goals	81
2	Notations and Terminology	91
2.1	Notational Conventions	91
2.2	Namespaces	91
2.3	Acronyms and Abbreviations	101
2.4	Terminology	101
2.5	Note on Examples	121
3	Message Protection Mechanisms	131
3.1	Message Security Model	131
3.2	Message Protection	131
3.3	Invalid or Missing Claims	141
3.4	Example	141
4	ID References	161
4.1	Id Attribute	161
4.2	Id Schema	161
5	Security Header	181
6	Security Tokens	211
6.1	Attaching Security Tokens	211
6.1.1	Processing Rules	211
6.1.2	Subject Confirmation	211
6.2	User Name Token	211
6.2.1	Usernames	211
6.3	Binary Security Tokens	221
6.3.1	Attaching Security Tokens	221
6.3.2	Encoding Binary Security Tokens	221
6.4	XML Tokens	231
6.5	EncryptedData Token	241
6.6	Identifying and Referencing Security Tokens	241
7	Token References	251
7.1	SecurityTokenReference Element	251
7.2	Direct References	271
7.3	Key Identifiers	281
7.4	Embedded References	301
7.5	ds:KeyInfo	311
7.6	Key Names	311
7.7	Encrypted Key reference	311
8	Signatures	331
8.1	Algorithms	331
8.2	Signing Messages	361
8.3	Signing Tokens	361
8.4	Signature Validation	391
8.5	Signature Confirmation	391
8.5.1	Response Generation Rules	401
8.5.2	Response Processing Rules	411
8.6	Example	411
9	Encryption	431
9.1	xenc:ReferenceList	431
9.2	xenc:EncryptedKey	441
9.3	Encrypted Header	451
9.4	Processing Rules	451
9.4.1	Encryption	451

... [1]

Deleted: 28

Deleted: June

170

1 Introduction

171

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

172

173

174

175

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

176

177

178

179

180

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

181

182

183

184

185

186

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

187

188

189

190

191

192

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

193

194

195

196

1.1 Goals and Requirements

197

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

198

199

200

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

201

202

203

204

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

205

206

207

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

208

209

210

211

212

The requirements to support secure message exchange are listed below.

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

11, October 2005
Page 7 of 74

Deleted: 28

Deleted: June

213 **1.1.1 Requirements**

214 The Web services security language must support a wide variety of security models. The
215 following list identifies the key driving requirements for this specification:
216 Multiple security token formats
217 Multiple trust domains
218 Multiple signature formats
219 Multiple encryption technologies
220 End-to-end message content security and not just transport-level security

← - - - Formatted: No bullets or numbering

221 **1.1.2 Non-Goals**

222 The following topics are outside the scope of this document:
223
224 Establishing a security context or authentication mechanisms.
225 Key derivation.
226 Advertisement and exchange of security policy.
227 How trust is established or determined.
228 Non-repudiation.
229

← - - - Formatted: No bullets or numbering

Deleted: 28
Deleted: June

230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247

248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270

2 Notations and Terminology

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

2.1 Notational Conventions

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

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

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

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

2.2 Namespaces

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

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

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

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

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

Deleted: 28
Deleted: June

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

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

275 The URLs provided for the *wsse* and *wsu* namespaces can be used to obtain the schema files.
 276

277
 278 | URI fragments defined in this document are relative to the following base URI unless otherwise
 279 stated:
 280 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0

Deleted: Most

281 2.3 Acronyms and Abbreviations

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

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

284 2.4 Terminology

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

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

11, October 2005
 Page 10 of 74

Deleted: 28

Deleted: June

286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325

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

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

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

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

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

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

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

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

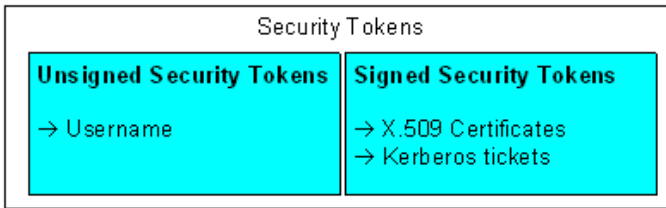
Message Confidentiality - *Message Confidentiality* is a property of the message and encryption is the mechanism by which this property of the message is provided.

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

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

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

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



Deleted:

Deleted: 28

Deleted: June

326

327
328
329
330
331
332

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

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

333 2.5 Note on Examples

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

337
338 Specifically, the examples are constrained to contain only mechanisms defined in this document.
339 The only reason for this is to avoid requiring the reader to consult other documents merely to
340 understand the examples. It is NOT intended to suggest that the mechanisms illustrated
341 represent best practice or are the strongest available to implement the security properties in
342 question. In particular, mechanisms defined in other Token Profiles are known to be stronger,
343 more efficient and/or generally superior to some of the mechanisms shown in the examples in this
344 document.
345

Deleted: 28

Deleted: June

346

3 Message Protection Mechanisms

347

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

348

349

350

the message could be modified or read by attacker or

351

an antagonist could send messages to a service that, while well-formed, lack appropriate security claims to warrant processing

352

353

an antagonist could alter a message to the service which being well formed causes the service to process and respond to the client for an incorrect request.

354

355

356

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

Formatted: No bullets or numbering

Deleted: antagonists

Formatted: Indent: Left: 0"

357

3.1 Message Security Model

358

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

359

360

361

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

362

363

364

365

366

367

368

369

370

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

371

372

373

374

375

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

376

377

378

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

379

380

381

3.2 Message Protection

382

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

383

384

385

386

Deleted: 28

Deleted: June

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

11, October 2005
Page 13 of 74

387 Message integrity is provided by XML Signature [XMLSIG] in conjunction with security tokens to
388 ensure that modifications to messages are detected. The integrity mechanisms are designed to
389 support multiple signatures, potentially by multiple SOAP actors/roles, and to be extensible to
390 support additional signature formats.

391
392 Message confidentiality leverages XML Encryption [XMLENC] in conjunction with security tokens
393 to keep portions of a SOAP message confidential. The encryption mechanisms are designed to
394 support additional encryption processes and operations by multiple SOAP actors/roles.
395

396 This document defines syntax and semantics of signatures within a <wsse:Security> element.

397 This document does not constrain any signature appearing outside of a <wsse:Security>
398 element.

Deleted: specify

399 3.3 Invalid or Missing Claims

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

407 3.4 Example

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

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

Deleted: 28

Deleted: June

```

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

```

Formatted: German (Germany)

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

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

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

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

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

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

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

Formatted: Indent: First line: 0"

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

Deleted: 28

Deleted: June

485 Lines (026) to (028) contain the body (payload) of the SOAP message.
486

Deleted: 28

Deleted: June

487 4 ID References

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

- 498 • Local ID attributes on XML Signature elements
- 499 • Local ID attributes on XML Encryption elements
- 500 • Global `wsu:Id` attributes (described below) on elements
- 501 • Profile specific defined identifiers

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

506 4.1 Id Attribute

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

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

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

523 4.2 Id Schema

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

527 The syntax for this attribute is as follows:

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

Formatted: Bulleted + Level: 1 +
Aligned at: 0.25" + Tab after: 0.5"
+ Indent at: 0.5"

Deleted: 28

Deleted: June

528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558

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

The following describes the attribute illustrated above:

.../@wsu:id

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

Two `wsu:id` attributes within an XML document MUST NOT have the same value. Implementations MAY rely on XML Schema validation to provide rudimentary enforcement for intra-document uniqueness. However, applications SHOULD NOT rely on schema validation alone to enforce uniqueness.

This specification does not specify how this attribute will be used and it is expected that other specifications MAY add additional semantics (or restrictions) for their usage of this attribute. The following example illustrates use of this attribute to identify an element:

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

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

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

Deleted: 28
Deleted: June

559

5 Security Header

560 The `<wsse:Security>` header block provides a mechanism for attaching security-related
561 information targeted at a specific recipient in the form of a SOAP actor/role. This may be either
562 the ultimate recipient of the message or an intermediary. Consequently, elements of this type
563 may be present multiple times in a SOAP message. An active intermediary on the message path
564 MAY add one or more new sub-elements to an existing `<wsse:Security>` header block if they
565 are targeted for its SOAP node or it MAY add one or more new headers for additional targets.
566

567 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted
568 for separate recipients. **A message MUST NOT have multiple `<wsse:Security>` header blocks
569 targeted (whether explicitly or implicitly) at the same recipient.** However, only one
570 `<wsse:Security>` header block MAY omit the `S11:actor` or `S12:role` attributes. Two
571 `<wsse:Security>` header blocks MUST NOT have the same value for `S11:actor` or
572 `S12:role`. Message security information targeted for different recipients MUST appear in
573 different `<wsse:Security>` header blocks. This is due to potential processing order issues
574 (e.g. due to possible header re-ordering). The `<wsse:Security>` header block without a
575 specified `S11:actor` or `S12:role` MAY be processed by anyone, but MUST NOT be removed
576 prior to the final destination or endpoint.
577

578 As elements are added to a `<wsse:Security>` header block, they SHOULD be prepended to
579 the existing elements. As such, the `<wsse:Security>` header block represents the signing and
580 encryption steps the message producer took to create the message. This prepending rule
581 ensures that the receiving application can process sub-elements in the order they appear in the
582 `<wsse:Security>` header block, because there will be no forward dependency among the sub-
583 elements. Note that this specification does not impose any specific order of processing the sub-
584 elements. The receiving application can use whatever order is required.
585

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

590
591
592
593
594
595
596
597
598
599
600

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

601
602 The following describes the attributes and elements listed in the example above:
603 `/wsse:Security`

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

11, October 2005
Page 19 of 74

Deleted: 28

Deleted: June

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

605

606 */wsse:Security/@S11:actor*

607 This attribute allows a specific SOAP 1.1 [SOAP11] actor to be identified. This attribute
608 is optional; however, no two instances of the header block may omit an actor or specify
609 the same actor.

Deleted: P

610

611 */wsse:Security/@S12:role*

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

615

616 */wsse:Security/@S11:mustUnderstand*

617 This SOAP 1.1 [SOAP11] attribute is used to indicate whether a header entry is
618 mandatory or optional for the recipient to process. The value of the mustUnderstand
619 attribute is either "1" or "0". The absence of the SOAP mustUnderstand attribute is
620 semantically equivalent to its presence with the value "0".

Deleted: P

621

622 */wsse:Security/@S12:mustUnderstand*

623 This SOAP 1.2 [SOAP12] attribute is used to indicate whether a header entry is
624 mandatory or optional for the recipient to process. The value of the mustUnderstand
625 attribute is either "true", "1", "false" or "0". The absence of the SOAP mustUnderstand
626 attribute is semantically equivalent to its presence with the value "false".

Deleted: or

627

628 */wsse:Security/{any}*

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

632

633 */wsse:Security/@{any}*

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

636

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

638

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

643

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

646

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

648 The receiver MUST generate a SOAP fault if does not implement the WSS: SOAP Message
649 Security specification corresponding to the namespace. Implementation means ability to interpret
650 the schema as well as follow the required processing rules specified in WSS: SOAP Message
651 Security.

Formatted: No bullets or numbering

Deleted: 28

Deleted: June

652 | The receiver MUST generate a fault if unable to interpret or process security tokens contained in
653 | the <wsse:Security> header block according to the corresponding WSS: SOAP Message
654 | Security token profiles.
655 | Receivers MAY ignore elements or extensions within the <wsse:Security> element, based on
656 | local security policy.

Deleted: 28

Deleted: June

657

6 Security Tokens

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

6.1 Attaching Security Tokens

661 This specification defines the `<wsse:Security>` header as a mechanism for conveying
662 security information with and about a SOAP message. This header is, by design, extensible to
663 support many types of security information.
664

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

6.1.1 Processing Rules

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

6.1.2 Subject Confirmation

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

6.2 User Name Token

6.2.1 Usernames

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

```
681  
682 <wsse:UsernameToken wsu:Id="...">  
683   <wsse:Username>...</wsse:Username>  
684 </wsse:UsernameToken>
```

685
686 The following describes the attributes and elements listed in the example above:

687 */wsse:UsernameToken*

688 This element is used to represent a claimed identity.

689
690 */wsse:UsernameToken/@wsu:Id*

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

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

11, October 2005
Page 22 of 74

Deleted: 28

Deleted: June

693
 694 `/wsse:UsernameToken/wsse:Username`
 695 This required element specifies the claimed identity.
 696
 697 `/wsse:UsernameToken/wsse:Username/@{any}`
 698 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 699 added to the `<wsse:Username>` element.
 700
 701 `/wsse:UsernameToken/{any}` ← --- Formatted: Indent: Left: 0.5"
 702 This is an extensibility mechanism to allow different (extensible) types of security
 703 information, based on a schema, to be passed. Unrecognized elements SHOULD cause
 704 a fault.
 705
 706 `/wsse:UsernameToken/@{any}`
 707 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 708 added to the `<wsse:UsernameToken>` element. Unrecognized attributes SHOULD
 709 cause a fault.
 710
 711 All compliant implementations MUST be able to process a `<wsse:UsernameToken>` ← --- Formatted: Indent: Left: 0.5"
 712 element.

713 The following illustrates the use of this:

```

714 <S11:Envelope xmlns:S11="..." xmlns:wsse="...">
715   <S11:Header>
716     ...
717     <wsse:Security>
718       <wsse:UsernameToken>
719         <wsse:Username>Zoe</wsse:Username>
720       </wsse:UsernameToken>
721     </wsse:Security>
722     ...
723   </S11:Header>
724   ...
725 </S11:Envelope>
726
727
  
```

728 6.3 Binary Security Tokens

729 6.3.1 Attaching Security Tokens

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

733 6.3.2 Encoding Binary Security Tokens

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

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

Deleted: 28

Deleted: June

738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769

770
771
772
773
774
775
776

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

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

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

`/wsse:BinarySecurityToken`

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

`/wsse:BinarySecurityToken/@wsu:Id`

An optional string label for this security token.

`/wsse:BinarySecurityToken/@ValueType`

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

`/wsse:BinarySecurityToken/@EncodingType`

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

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

Deleted: (note that the URI fragments are relative to the URI for this specification)

`/wsse:BinarySecurityToken/@{any}`

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

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

Deleted: 28
Deleted: June

777 **6.4 XML Tokens**

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

780 **6.5 EncryptedData Token**

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

786
787 It should be noted that token references are not made to the <xenc:EncryptedData> element,
788 but instead to the token represented by the clear-text, once the <xenc:EncryptedData>
789 element has been processed (decrypted). Such references utilize the token profile for the
790 contained token. i.e., <xenc:EncryptedData> SHOULD NOT include an XML Id for
791 referencing the contained security token.

792
793 All <xenc:EncryptedData> tokens SHOULD either have an embedded encryption key or
794 should be referenced by a separate encryption key.

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

797 **6.6 Identifying and Referencing Security Tokens**

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

803

7 Token References

804

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

805

806

7.1 SecurityTokenReference Element

807

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

808

809

810

811

812

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

813

814

815

816

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

819

The following illustrates the syntax of this element:

820

821

822

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

823

824

825

The following describes the elements defined above:

826

827

/wsse:SecurityTokenReference

828

This element provides a reference to a security token.

829

830

/wsse:SecurityTokenReference/@wsu:Id

831

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

832

833

834

835

836

/wsse:SecurityTokenReference/@wsse11:TokenType

837

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

838

This specification recommends that token specific profiles define appropriate token type identifying URI values, and that these same profiles require that these values be specified in the profile defined reference forms.

839

840

841

842

Deleted: containing element

Deleted: >¶

Deleted: . . .

Formatted: Tabs: Not at 0.64" + 1.27" + 1.91" + 2.54" + 3.18" + 3.82" + 4.45" + 5.09" + 5.73" + 6.36" + 7" + 7.63" + 8.27" + 8.91" + 9.54" + 10.18"

Deleted: 28

Deleted: June

843 | When a `wss11:TokenType` attribute is specified in conjunction with a
 844 | `wsse:KeyIdentifier/@ValueType` attribute or a `wsse:Reference/@ValueType`
 845 | attribute that indicates the type of the referenced token, the security token type identified
 846 | by the `wss11:TokenType` attribute MUST be consistent with the security token type
 847 | identified by the `wsse:ValueType` attribute.
 848 |

URI	Description
http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-soap-message-security-1.1#EncryptedKey	A token type of an <code><xenc:EncryptedKey></code>

- Formatted: Font: Courier New
- Formatted: Font: Courier New
- Deleted: TokenType
- Formatted: Font: Courier New
- Formatted: Font: Courier New
- Deleted: v
- Formatted: Font: Courier New

849 | `/wsse:SecurityTokenReference/@wsse:Usage`
 850 | This optional attribute is used to type the usage of the
 851 | `<wsse:SecurityTokenReference>`. Usages are specified using URIs and multiple
 852 | usages MAY be specified using XML list semantics. No usages are defined by this
 853 | specification.
 854 |
 855 | `/wsse:SecurityTokenReference/{any}`
 856 | This is an extensibility mechanism to allow different (extensible) types of security
 857 | references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
 858 | fault.
 859 |
 860 | `/wsse:SecurityTokenReference/@{any}`
 861 | This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 862 | added to the header. Unrecognized attributes SHOULD cause a fault.
 863 |
 864 |

- Deleted: ¶

865 | All compliant implementations MUST be able to process a
 866 | `<wsse:SecurityTokenReference>` element.
 867 |
 868 | This element can also be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to
 869 | retrieve the key information from a security token placed somewhere else. In particular, it is
 870 | RECOMMENDED, when using XML Signature and XML Encryption, that a
 871 | `<wsse:SecurityTokenReference>` element be placed inside a `<ds:KeyInfo>` to reference
 872 | the security token used for the signature or encryption.
 873 |
 874 | There are several challenges that implementations face when trying to interoperate. Processing
 875 | the IDs and references requires the recipient to *understand* the schema. This may be an
 876 | expensive task and in the general case impossible as there is no way to know the "schema
 877 | location" for a specific namespace URI. As well, the primary goal of a reference is to uniquely
 878 | identify the desired token. ID references are, by definition, unique by XML. However, other
 879 | mechanisms such as "principal name" are not required to be unique and therefore such
 880 | references may be not unique.
 881 |

882 | [This specification allows for the use of multiple reference mechanisms within a single](#)
 883 | [SecurityTokenReference. When multiple references are present in a given](#)

- Deleted: 28
- Deleted: June

884 [SecurityTokenReference, they MUST resolve to a single token in common. Specific token](#)
885 [profiles SHOULD define the reference mechanisms to be used.](#)
886

887 The following list provides a list of the specific reference mechanisms defined in WSS: SOAP
888 Message Security in preferred order (i.e., most specific to least specific):
889

- 890 • **Direct References** – This allows references to included tokens using URI fragments and
891 external tokens using full URIs.
- 892 • **Key Identifiers** – This allows tokens to be referenced using an opaque value that
893 represents the token (defined by token type/profile).
- 894 • **Key Names** – This allows tokens to be referenced using a string that matches an identity
895 assertion within the security token. This is a subset match and may result in multiple
896 security tokens that match the specified name.
- 897 • **Embedded References** - This allows tokens to be embedded (as opposed to a pointer
898 to a token that resides elsewhere).

899 7.2 Direct References

900 The <wsse:Reference> element provides an extensible mechanism for directly referencing
901 security tokens using URIs.
902

903 The following illustrates the syntax of this element:

```
904 <wsse:SecurityTokenReference wsu:Id="...">  
905   <wsse:Reference URI="..." ValueType="..." />  
906 </wsse:SecurityTokenReference>
```

907
908 The following describes the elements defined above:

909
910 */wsse:SecurityTokenReference/wsse:Reference*

911 This element is used to identify an abstract URI location for locating a security token.
912
913

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

915 This optional attribute specifies an abstract URI for where to find a security token. If a
916 fragment is specified, then it indicates the local ID of the token being referenced.
917

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

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

924
925 In this version of the specification the use of this attribute to identify the type of the
926 referenced security token is deprecated. Profiles which require or recommend the use of
927 this attribute to identify the type of the referenced security token SHOULD evolve to
928 require or recommend the use of the

929 `wsse:SecurityTokenReference/@wsse11:TokenType` attribute to identify the type
930 of the referenced token.

Deleted: how it SHALL

Deleted: be

Deleted: ed.

Deleted:

Deleted: T

Deleted: 28

Deleted: June

931
932 /wsse:SecurityTokenReference/wsse:Reference/{any}
933 This is an extensibility mechanism to allow different (extensible) types of security
934 references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
935 fault.

936
937 /wsse:SecurityTokenReference/wsse:Reference/@{any}
938 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
939 added to the header. Unrecognized attributes SHOULD cause a fault.

940
941 The following illustrates the use of this element:

```
942 <wsse:SecurityTokenReference  
943     xmlns:wsse="...">  
944   <wsse:Reference  
945     URI="http://www.fabrikaml23.com/tokens/Zoe"/>  
946 </wsse:SecurityTokenReference>
```

948 7.3 Key Identifiers

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

Deleted: to use
Deleted: bi

955
956 The <wsse:KeyIdentifier> element SHALL is placed in the
957 <wsse:SecurityTokenReference> element to reference a token using an identifier. This
958 element SHOULD be used for all key identifiers.

Deleted: be

959
960 The processing model assumes that the key identifier for a security token is constant.
961 Consequently, processing a key identifier involves simply looking for a security token whose key
962 identifier matches the specified constant. The <wsse:KeyIdentifier> element is only allowed
963 inside a <wsse:SecurityTokenReference> element

Deleted: is
Deleted: a
Deleted: given

964 The following is an overview of the syntax:

```
965 <wsse:SecurityTokenReference>  
966   <wsse:KeyIdentifier wsu:Id="..."  
967     ValueType="..."  
968     EncodingType="...">  
969     ...  
970   </wsse:KeyIdentifier>  
971 </wsse:SecurityTokenReference>
```

972
973 The following describes the attributes and elements listed in the example above:

974 /wsse:SecurityTokenReference/wsse:KeyIdentifier
975 This element is used to include a binary-encoded key identifier.

Deleted: 28
Deleted: June

979 /wsse:SecurityTokenReference/wsse:KeyIdentifier/@wsu:Id
980 An optional string label for this identifier.

981 /wsse:SecurityTokenReference/wsse:KeyIdentifier/@ValueType

982 The optional ValueType attribute is used to indicate the type of KeyIdentifier being
983 used. This specification defines one ValueType that can be applied to all token types.
984 Each specific token profile specifies the KeyIdentifier types that may be used to
985 refer to tokens of that type. It also specifies the critical semantics of the identifier, such as
986 whether the KeyIdentifier is unique to the key or the token. If no value is specified
987 then the key identifier will be interpreted in an application-specific manner. This URI
988 fragment is relative to a base URI of
989 http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-soap-message-security-1.1
990
991

Formatted: Indent: Left: 0"

URI	Description
http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-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 KeyIdentifier MUST be the SHA1 of the raw octets which would be encoded within the security token element were it to be included. <u>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.</u>
http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-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 EncryptedKey, the value obtained from the token MAY be used. If the token does not contain a representation of a EncryptedKey, then the value of the KeyIdentifier MUST be the SHA1 of the raw octets which would be encoded within the security token element were it to be

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Deleted: 28

Deleted: June

included.

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Deleted: used (Note that URI fragments are relative to this document's URI):

992
993
994
995
996
997
998
999
1000

/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

1001
1002
1003
1004

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

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

7.4 Embedded References

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

The following is an overview of the syntax:

1011
1012
1013
1014
1015
1016

```

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

```

1017
1018

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

1019
1020
1021
1022
1023

/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).

1024
1025
1026

/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.

1027
1028
1029
1030

/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.

1031
1032

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

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

11, October 2005
Page 31 of 74

Deleted: 28

Deleted: June

1033 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
1034 added. Unrecognized attributes SHOULD cause a fault.

1035

The following example illustrates embedding a SAML assertion:

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

```
<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>
```

1054

7.5 ds:KeyInfo

1055

1056

1057

1058

1059

1060

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.

1061

1062

1063

1064

1065

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

1066

7.6 Key Names

1067

1068

1069

1070

1071

1072

1073

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
```

1074

7.7 Encrypted Key reference

1075

1076

1077

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

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

11, October 2005
Page 32 of 74

Deleted: 28

Deleted: June

1078 | The ~~EncryptedKey~~ MAY be used to perform other cryptographic operations within the same
1079 | message, such as signatures. The EncryptedKey MAY also be used for performing
1080 | cryptographic operations in subsequent messages exchanged by the two parties. Two
1081 | mechanisms are defined for referencing the EncryptedKey.

Deleted: EncryptedKey

1082 |
1083 | When referencing the EncryptedKey within the same message that contains the
1084 | <xenc:EncryptedKey> element, the <ds:KeyInfo> element of the referencing construct
1085 | MUST contain a <wsse:SecurityTokenReference>. The
1086 | <wsse:SecurityTokenReference> element MUST contain a <wsse:Reference> element.

1087 |
1088 | The URI attribute value of the <wsse:Reference> element MUST be set to the value of the ID
1089 | attribute of the referenced <xenc:EncryptedKey> element that contains the EncryptedKey.

1090 | When referencing the EncryptedKey in a message that does not contain the
1091 | <xenc:EncryptedKey> element, the <ds:KeyInfo> element of the referencing construct
1092 | MUST contain a <wsse:SecurityTokenReference>. The

Formatted: Font: Courier New

1093 | <wsse:SecurityTokenReference> element MUST contain a <wsse:KeyIdentifier>
1094 | element. The EncodingType attribute SHOULD be set to #Base64Binary. Other encoding
1095 | types MAY be specified if agreed on by all parties. The ~~wsse1:TokenType~~ attribute MUST be
1096 | set to

Deleted: ValueType

1097 | http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-soap-message-
1098 | security-1.1#EncryptedKey. The identifier for a <xenc:EncryptedKey> token is defined
1099 | as the SHA1 of the raw (pre-base64 encoding) octets specified in the <xenc:CipherValue>
1100 | element of the referenced <xenc:EncryptedKey> token. This value is encoded as indicated in
1101 | the KeyIdentifier reference. The ~~wsse:ValueType~~ attribute of <wsse:KeyIdentifier>
1102 | MUST be set to http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-
1103 | soap-message-security-1.1#EncryptedKeySHA1

Deleted: V

Deleted: 28

Deleted: June

1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135

1136
1137
1138
1139
1140
1141

8 Signatures

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

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

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

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

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

8.1 Algorithms

This specification builds on XML Signature and therefore has the same algorithm requirements as those specified in the XML Signature specification. The following table outlines additional algorithms that are strongly RECOMMENDED by this specification:

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

Deleted: 28
Deleted: June

1142
1143
1144

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/

1145
1146
1147
1148
1149
1150
1151
1152
1153
1154

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]
.
.

Formatted: Left
Formatted: Left, Line spacing: single
Formatted: Left

1155
1156
1157
1158
1159
1160

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]
.
.

Formatted: Left
Formatted: Left, Line spacing: single
Formatted: Left

1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171

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

There are two problems to be avoided. On the one hand, XML allows documents to be changed in various ways and still be considered equivalent. For example, duplicate namespace declarations can be removed or created. As a result, XML tools make these kinds of changes

Deleted: 28
Deleted: June

1172 freely when processing XML. Therefore, it is vital that these equivalent forms match the same
1173 signature.

1174
1175 On the other hand, if the signature simply covers something like `xx:foo`, its meaning may change
1176 if `xx` is redefined. In this case the signature does not prevent tampering. It might be thought that
1177 the problem could be solved by expanding all the values in line. Unfortunately, there are
1178 mechanisms like XPATH which consider `xx="http://example.com/"`; to be different from
1179 `yy="http://example.com/"`; even though both `xx` and `yy` are bound to the same namespace.
1180 The fundamental difference between the Inclusive and Exclusive Canonicalization is the
1181 namespace declarations which are placed in the output. Inclusive Canonicalization copies all the
1182 declarations that are currently in force, even if they are defined outside of the scope of the
1183 signature. It also copies any `xml: attributes` that are in force, such as `xml: lang` or `xml: base`.
1184 This guarantees that all the declarations you might make use of will be unambiguously specified.
1185 The problem with this is that if the signed XML is moved into another XML document which has
1186 other declarations, the Inclusive Canonicalization will copy them and the signature will be invalid.
1187 This can even happen if you simply add an attribute in a different namespace to the surrounding
1188 context.

1189
1190 Exclusive Canonicalization tries to figure out what namespaces you are actually using and just
1191 copies those. Specifically, it copies the ones that are "visibly used", which means the ones that
1192 are a part of the XML syntax. However, it does not look into attribute values or element content,
1193 so the namespace declarations required to process these are not copied. For example
1194 if you had an attribute like `xx:foo="yy:bar"` it would copy the declaration for `xx`, but not `yy`. (This
1195 can even happen without your knowledge because XML processing tools might add `xsi: type` if
1196 you use a schema subtype.) It also does not copy the `xml: attributes` that are declared outside the
1197 scope of the signature.

Deleted: will

1198
1199 Exclusive Canonicalization allows you to create a list of the namespaces that must be declared,
1200 so that it will pick up the declarations for the ones that are not visibly used. The only problem is
1201 that the software doing the signing must know what they are. In a typical SOAP software
1202 environment, the security code will typically be unaware of all the namespaces being used by the
1203 application in the message body that it is signing.

1204
1205 Exclusive Canonicalization is useful when you have a signed XML document that you wish to
1206 insert into other XML documents. A good example is a signed SAML assertion which might be
1207 inserted as a XML Token in the security header of various SOAP messages. The Issuer who
1208 signs the assertion will be aware of the namespaces being used and able to construct the list.
1209 The use of Exclusive Canonicalization will insure the signature verifies correctly every time.
1210 Inclusive Canonicalization is useful in the typical case of signing part or all of the SOAP body in
1211 accordance with this specification. This will insure all the declarations fall under the signature,
1212 even though the code is unaware of what namespaces are being used. At the same time, it is
1213 less likely that the signed data (and signature element) will be inserted in some other XML
1214 document. Even if this is desired, it still may not be feasible for other reasons, for example there
1215 may be `Id`'s with the same value defined in both XML documents.

1216
1217 In other situations it will be necessary to study the requirements of the application and the
1218 detailed operation of the canonicalization methods to determine which is appropriate.
1219 This section is non-normative.

Deleted: 28

Deleted: June

1220

8.2 Signing Messages

1221

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

1222

1223

1224

1225

1226

1227

SOAP applications MUST satisfy the following conditions:

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

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

Deleted: subsequent

1253

8.3 Signing Tokens

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

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

Deleted: 28

Deleted: June

1264 | This transform is specified by the URI #STR-Transform, and when applied to a
1265 <wsse:SecurityTokenReference> element it means that the output is the token referenced
1266 by the <wsse:SecurityTokenReference> element not the element itself.

Deleted: (Note that URI fragments are relative to this document's URI)

1267
1268 As an overview the processing model is to echo the input to the transform except when a
1269 <wsse:SecurityTokenReference> element is encountered. When one is found, the element
1270 is not echoed, but instead, it is used to locate the token(s) matching the criteria and rules defined
1271 by the <wsse:SecurityTokenReference> element and echo it (them) to the output.
1272 Consequently, the output of the transformation is the resultant sequence representing the input
1273 with any <wsse:SecurityTokenReference> elements replaced by the referenced security
1274 token(s) matched.

1275
1276 The following illustrates an example of this transformation which references a token contained
1277 within the message envelope:

```
1278 ...  
1279 <wsse:SecurityTokenReference wsu:Id="Str1">  
1280 ...  
1281 </wsse:SecurityTokenReference>  
1282 ...  
1283 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
1284   <ds:SignedInfo>  
1285     ...  
1286     <ds:Reference URI="#Str1">  
1287       <ds:Transforms>  
1288         <ds:Transform  
1289           Algorithm="...#STR-Transform">  
1290             <wsse:TransformationParameters>  
1291               <ds:CanonicalizationMethod  
1292                 Algorithm="http://www.w3.org/TR/2001/REC-xml-  
1293 c14n-20010315" />  
1294             </wsse:TransformationParameters>  
1295           </ds:Transform>  
1296           <ds:DigestMethod Algorithm=  
1297             "http://www.w3.org/2000/09/xmldsig#sha1" />  
1298           <ds:DigestValue>...</ds:DigestValue>  
1299         </ds:Reference>  
1300       </ds:SignedInfo>  
1301       <ds:SignatureValue></ds:SignatureValue>  
1302     </ds:Signature>  
1303   </ds:Signature>  
1304   ...  
1305
```

1306 The following describes the attributes and elements listed in the example above:

1307 /wsse:TransformationParameters

1308 This element is used to wrap parameters for a transformation allows elements even from
1309 the XML Signature namespace.

1310 /wsse:TransformationParameters/ds:Canonicalization

1311 This specifies the canonicalization algorithm to apply to the selected data.

Deleted: canonicalization

Deleted: 28

Deleted: June

1315 /wsse:TransformationParameters/{any}
1316 This is an extensibility mechanism to allow different (extensible) parameters to be
1317 specified in the future. Unrecognized parameters SHOULD cause a fault.
1318

1319 /wsse:TransformationParameters/@{any}
1320 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
1321 added to the element in the future. Unrecognized attributes SHOULD cause a fault.
1322

1323 The following is a detailed specification of the transformation. The algorithm is identified by the
1324 URI: #STR-Transform.

1325
1326 Transform Input:

- 1327 • The input is a node set. If the input is an octet stream, then it is automatically parsed; cf.
1328 XML Digital Signature [XMLSIG].

1329 Transform Output:

- 1330 • The output is an octet stream.

1331 Syntax:

- 1332 • The transform takes a single mandatory parameter, a
1333 <ds:CanonicalizationMethod> element, which is used to serialize the **output** node
1334 set. Note, however, that the output may not be strictly in canonical form, per the
1335 canonicalization algorithm; however, the output is canonical, in the sense that it is
1336 unambiguous. However, because of syntax requirements in the XML Signature
1337 definition, this parameter MUST be wrapped in a
1338 <wsse:TransformationParameters> element.

Deleted: input

1339
1340 Processing Rules:

- 1341 • Let N be the input node set.
- 1342 • Let R be the set of all <wsse:SecurityTokenReference> elements in N.
- 1343 • For each Ri in R, let Di be the result of dereferencing Ri.
- 1344 • If Di cannot be determined, then the transform MUST signal a failure.
- 1345 • If Di is an XML security token (e.g., a SAML assertion or a
1346 <wsse:BinarySecurityToken> element), then let Ri' be Di. Otherwise, Di is a raw
1347 binary security token; i.e., an octet stream. In this case, let Ri' be a node set consisting of
1348 a <wsse:BinarySecurityToken> element, utilizing the same namespace prefix as
1349 the <wsse:SecurityTokenReference> element Ri, with no EncodingType attribute,
1350 a ValueType attribute identifying the content of the security token, and text content
1351 consisting of the binary-encoded security token, with no white space.
- 1352 • Finally, employ the canonicalization method specified as a parameter to the transform to
1353 serialize N to produce the octet stream output of this transform; but, in place of any
1354 dereferenced <wsse:SecurityTokenReference> element Ri and its descendants,
1355 process the dereferenced node set Ri' instead. During this step, canonicalization of the
1356 replacement node set MUST be augmented as follows:
 - 1357 o Note: A namespace declaration xmlns="" MUST be emitted with every apex
1358 element that has no namespace node declaring a value for the default
1359 namespace; cf. XML Decryption Transform.

1360
1361 Signing a SecurityTokenReference (STR) provides authentication and integrity protection
1362 of only the STR and not the referenced security token (ST). If signing the ST is the

Deleted: 28

Deleted: June

1363
1364
1365
1366
1367
1368
1369

intended behavior, the STR Dereference Transform (STRDT) may be used which replaces the STR with the ST for digest computation, effectively protecting the ST and not the STR. If protecting both the ST and the STR is desired, you may sign the STR twice, once using the STRDT and once not using the STRDT.

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

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

1370

8.4 Signature Validation

1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384

The validation of a `<ds:Signature>` element inside an `<wsse:Security>` header block **MUST** fail if:
the syntax of the content of the element does not conform to this specification, or
the validation of the signature contained in the element fails according to the core validation of the XML Signature specification [XMLSIG], or
the application applying its own validation policy rejects the message for some reason (e.g., the signature is created by an untrusted key – verifying the previous two steps only performs cryptographic validation of the signature).

Deleted: SHALL

Formatted: No bullets or numbering

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

Formatted: Indent: Left: 0"

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

1385

8.5 Signature Confirmation

1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399

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

Compliant responder implementations that support signature confirmation, **MUST** include a `<wsse11:SignatureConfirmation>` element inside the `<wsse:Security>` header of the associated response message for every `<ds:Signature>` element that is a direct child of the `<wsse:Security>` header block in the originating message. The responder **MUST** include the

Deleted: 28

Deleted: June

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

11, October 2005
Page 40 of 74

1400 contents of the <ds:SignatureValue> element of the request signature as the value of the
1401 @Value attribute of the <wsse11:SignatureConfirmation> element. The
1402 <wsse11:SignatureConfirmation> element MUST be included in the message signature of
1403 the associated response message.

1404
1405 If the associated originating signature is received in encrypted form then the corresponding
1406 <wsse11:SignatureConfirmation> element SHOULD be encrypted to protect the original
1407 signature and keys.

1408
1409 The schema outline for this element is as follows:

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

1410
1411 /wsse11:SignatureConfirmation

1412 This element indicates that the responder has processed the signature in the request.
1413 When this element is not present in a response the initiator SHOULD interpret that the
1414 responder is not compliant with this functionality.

1415
1416 /wsse11:SignatureConfirmation/@wsu:Id

1417 Identifier to be used when referencing this element in the SignedInfo reference list of the
1418 signature of the associated response message. This attribute MUST be present so that
1419 un-ambiguous references can be made to this <wsse11:SignatureConfirmation>
1420 element.

1421
1422 /wsse11:SignatureConfirmation/@Value

1423 This optional attribute contains the contents of a <ds:SignatureValue> copied from
1424 the associated request. If the request was not signed, then this attribute MUST NOT be
1425 present. If this attribute is specified with an empty value, the initiator SHOULD interpret
1426 this as incorrect behavior and process accordingly. When this attribute is not present, the
1427 initiator SHOULD interpret this to mean that the response is based on a request that was
1428 not signed.

1429 8.5.1 Response Generation Rules

1430 Conformant responders MUST include at least one <wsse11:SignatureConfirmation>
1431 element in the <wsse:Security> header in any response(s) associated with requests. That is,
1432 the normal messaging patterns are not altered.
1433 For every response message generated, the responder MUST include a
1434 <wsse11:SignatureConfirmation> element for every <ds:Signature> element it
1435 processed from the original request message. The Value attribute MUST be set to the exact
1436 value of the <ds:SignatureValue> element of the corresponding <ds:Signature> element.
1437 If no <ds:Signature> elements are present in the original request message, the responder
1438 MUST include exactly one <wsse11:SignatureConfirmation> element. The Value attribute
1439 of the <wsse11:SignatureConfirmation> element MUST NOT be present. The responder
1440 MUST include all <wsse11:SignatureConfirmation> elements in the message signature of
1441 the response message(s). If the <ds:Signature> element corresponding to a
1442 <wsse11:SignatureConfirmation> element was encrypted in the original request message,
1443 the <wsse11:SignatureConfirmation> element SHOULD be encrypted for the recipient of
1444 the response message(s).

Deleted: If the responder does not comply with this specification, it MUST NOT include any <wsse11:SignatureConfirmation> elements in response messages it generates

Deleted: If

Formatted: Text Char1,t Char,t Char1, Font: (Default) Helvetica, (Asian) MS Mincho, Font color: Auto, (Asian) Japanese

Deleted: the responder complies with this specification, it MUST include at least one <wsse11:SignatureConfirmation>

Deleted: 28

Deleted: June

1445

1446 8.5.2 Response Processing Rules

1447 The signature validation shall additionally adhere to the following processing guidelines, if the
1448 initiator desires signature confirmation:

- 1449 • If a response message does not contain a <wssell:SignatureConfirmation>
1450 element inside the <wsse:Security> header, the initiator SHOULD reject the response
1451 message.
- 1452 • If a response message does contain a <wssell:SignatureConfirmation> element
1453 inside the <wsse:Security> header but @Value attribute is not present on
1454 <wssell:SignatureConfirmation> element, and the associated request message
1455 did include a <ds:Signature> element, the initiator SHOULD reject the response
1456 message.
- 1457 • If a response message does contain a <wssell:SignatureConfirmation> element
1458 inside the <wsse:Security> header and the @Value attribute is present on the
1459 <wssell:SignatureConfirmation> element, but the associated request did not
1460 include a <ds:Signature> element, the initiator SHOULD reject the response
1461 message.
- 1462 • If a response message does contain a <wssell:SignatureConfirmation> element
1463 inside the <wsse:Security> header, and the associated request message did include
1464 a <ds:Signature> element and the @Value attribute is present but does not match the
1465 stored signature value of the associated request message, the initiator SHOULD reject
1466 the response message.
- 1467 • If a response message does not contain a <wssell:SignatureConfirmation>
1468 element inside the <wsse:Security> header corresponding to each
1469 <ds:Signature> element or if the @Value attribute present does not match the stored
1470 signature values of the associated request message, the initiator SHOULD reject the
1471 response message.

Deleted: a

1472 8.6 Example

1473 The following sample message illustrates the use of integrity and security tokens. For this
1474 example, only the message body is signed.

```

1475 <?xml version="1.0" encoding="utf-8"?>
1476 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1477 xmlns:ds="...">
1478   <S11:Header>
1479     <wsse:Security>
1480       <wsse:BinarySecurityToken
1481         ValueType="...#X509v3"
1482         EncodingType="...#Base64Binary"
1483         wsu:Id="X509Token">
1484           MIIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1485       </wsse:BinarySecurityToken>
1486     <ds:Signature>
1487       <ds:SignedInfo>
1488         <ds:CanonicalizationMethod Algorithm=
1489
```

Deleted: 28

Deleted: June

```
1490         "http://www.w3.org/2001/10/xml-exc-c14n#" />
1491     <ds:SignatureMethod Algorithm=
1492         "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
1493     <ds:Reference URI="#myBody">
1494         <ds:Transforms>
1495             <ds:Transform Algorithm=
1496                 "http://www.w3.org/2001/10/xml-exc-c14n#" />
1497         </ds:Transforms>
1498         <ds:DigestMethod Algorithm=
1499             "http://www.w3.org/2000/09/xmldsig#sha1" />
1500         <ds:DigestValue>EULddytSol...</ds:DigestValue>
1501     </ds:Reference>
1502 </ds:SignedInfo>
1503 <ds:SignatureValue>
1504     BL8jdfToEb11/vXcMZNNjPOV...
1505 </ds:SignatureValue>
1506 <ds:KeyInfo>
1507     <wsse:SecurityTokenReference>
1508         <wsse:Reference URI="#X509Token" />
1509     </wsse:SecurityTokenReference>
1510 </ds:KeyInfo>
1511 </ds:Signature>
1512 </wsse:Security>
1513 </S11:Header>
1514 <S11:Body wsu:Id="myBody">
1515     <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
1516         QQQ
1517     </tru:StockSymbol>
1518 </S11:Body>
1519 </S11:Envelope>
```

Deleted: 28

Deleted: June

1520

9 Encryption

1521

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

1522

1523

1524

1525

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

1526

1527

1528

1529

1530

1531

1532

1533

1534

1535

1536

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

1537

1538

1539

1540

1541

1542

9.1 xenc:ReferenceList

1543

The `<xenc:ReferenceList>` element from XML Encryption [XMLENC] MAY be used to create a manifest of encrypted portion(s), which are expressed as `<xenc:EncryptedData>` elements within the envelope. 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 `<xenc:DataReference>` elements inside one or more `<xenc:ReferenceList>` element.

1544

1545

1546

1547

1548

1549

1550

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

1551

1552

1553

1554

1555

1556

1557

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

1558

1559

1560

1561

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

11, October 2005
Page 44 of 74

Deleted: 28

Deleted: June

```

1562 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1563 xmlns:ds="..." xmlns:xenc="...">
1564   <S11:Header>
1565     <wsse:Security>
1566       <xenc:ReferenceList>
1567         <xenc:DataReference URI="#bodyID" />
1568       </xenc:ReferenceList>
1569     </wsse:Security>
1570   </S11:Header>
1571   <S11:Body>
1572     <xenc:EncryptedData Id="bodyID">
1573       <ds:KeyInfo>
1574         <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
1575       </ds:KeyInfo>
1576       <xenc:CipherData>
1577         <xenc:CipherValue>...</xenc:CipherValue>
1578       </xenc:CipherData>
1579     </xenc:EncryptedData>
1580   </S11:Body>
1581 </S11:Envelope>

```

9.2 xenc:EncryptedKey

1583 When the encryption step involves encrypting elements or element contents within a SOAP
 1584 envelope with a symmetric key, which is in turn to be encrypted by the recipient's key and
 1585 embedded in the message, `<xenc:EncryptedKey>` MAY be used for carrying such an
 1586 encrypted key. This sub-element MAY contain a manifest, that is, an `<xenc:ReferenceList>`
 1587 `<xenc:EncryptedKey>` element, that lists the portions to be decrypted with this key. The manifest MAY appear outside
 1588 the `<xenc:EncryptedKey>` provided that the corresponding `<xenc:EncryptedData`
 1589 `<xenc:EncryptedData>` elements contain `<xenc:KeyInfo>` elements that reference the `<xenc:EncryptedKey>`. An element or
 1590 element content to be encrypted by this encryption step MUST be replaced by a corresponding
 1591 `<xenc:EncryptedData>` according to XML Encryption. All the `<xenc:EncryptedData>`
 1592 elements created by this encryption step SHOULD be listed in the `<xenc:ReferenceList>`
 1593 element inside this sub-element.

1594 This construct is useful when encryption is done by a randomly generated symmetric key that is
 1595 in turn encrypted by the recipient's public key. The following illustrates the use of this element:

```

1597 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1598 xmlns:ds="..." xmlns:xenc="...">
1599   <S11:Header>
1600     <wsse:Security>
1601       <xenc:EncryptedKey>
1602         ...
1603       <ds:KeyInfo>
1604         <wsse:SecurityTokenReference>
1605           <ds:X509IssuerSerial>
1606             <ds:X509IssuerName>
1607               DC=ACMECorp, DC=com
1608             </ds:X509IssuerName>
1609           <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
1610         </wsse:SecurityTokenReference>
1611       </ds:KeyInfo>

```

Formatted: Tabs: Not at 0.64" + 1.27" + 1.91" + 2.54" + 3.18" + 3.82" + 4.45" + 5.09" + 5.73" + 6.36" + 7" + 7.63" + 8.27" + 8.91" + 9.54" + 10.18"

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: (Default) Courier New, 10 pt

Formatted: Font: (Default) Helvetica, 10 pt

Formatted: Font: (Default) Courier New, 10 pt

Deleted: This sub-element SHOULD have a manifest, that is, an `<xenc:ReferenceList>` element, in order for the recipient to know the portions to be decrypted with this key

Deleted:

Formatted: Font: (Default) Helvetica, 10 pt

Deleted: 28

Deleted: June

1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627

```
        </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        ...
    </xenc:EncryptedKey>
    ...
</wsse:Security>
</S11:Header>
<S11:Body>
  <xenc:EncryptedData Id="bodyID">
    <xenc:CipherData>
      <xenc:CipherValue>...</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</S11:Body>
</S11:Envelope>
```

1628 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in
1629 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that
1630 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

1631 9.3 Encrypted Header

1632 In order to be compliant with SOAP mustUnderstand processing guidelines and to prevent
1633 disclosure of information contained in attributes on a SOAP header block, this specification
1634 introduces an `<wsse11:EncryptedHeader>` element. This element contains exactly one
1635 `<xenc:EncryptedData>` element. This specification RECOMMENDS the use of
1636 `<wsse11:EncryptedHeader>` element for encrypting SOAP header blocks.

1637 9.4 Processing Rules

1638 Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the
1639 XML Encryption specification. An encrypted SOAP envelope MUST still be a valid SOAP
1640 envelope. The message creator MUST NOT encrypt the `<S11:Header>`, `<S12:Header>`,
1641 `<S11:Envelope>`, `<S12:Envelope>`, or `<S11:Body>`, `<S12:Body>` elements but MAY
1642 encrypt child elements of either the `<S11:Header>`, `<S12:Header>` and `<S11:Body>` or
1643 `<S12:Body>` elements. Multiple steps of encryption MAY be added into a single
1644 `<wsse:Security>` header block if they are targeted for the same recipient.

Formatted: Font: Courier New

1645
1646 When an element or element content inside a SOAP envelope (e.g. the contents of the
1647 `<S11:Body>` or `<S12:Body>` elements) are to be encrypted, it MUST be replaced by an
1648 `<xenc:EncryptedData>`, according to XML Encryption and it SHOULD be referenced from the
1649 `<xenc:ReferenceList>` element created by this encryption step. If the target of reference is
1650 an `EncryptedHeader` as defined in section 9.3 above, see processing rules defined in section
1651 9.5.3 Encryption using EncryptedHeader and section 9.5.4 Decryption of EncryptedHeader
1652 below.

Formatted: Font: Courier New

1653 9.4.1 Encryption

1654 The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1655 this specification are listed below (note that use of `<xenc:ReferenceList>` is
WSS: SOAP Message Security (WS-Security 2004)
Copyright © OASIS Open 2002-2005. All Rights Reserved.

Deleted: 28

Deleted: June

1656 | RECOMMENDED. Additionally, if the target of encryption is a SOAP header, processing rules
1657 | defined in section 9.5.3 SHOULD be used).
1658 | Create a new SOAP envelope.
1659 | Create a <wsse:Security> header
1660 | When an <xenc:EncryptedKey> is used, create a <xenc:EncryptedKey> sub-element of
1661 | the <wsse:Security> element. This <xenc:EncryptedKey> sub-element SHOULD contain
1662 | an <xenc:ReferenceList> sub-element, containing a <xenc:DataReference> to each
1663 | <xenc:EncryptedData> element that was encrypted using that key.
1664 | Locate data items to be encrypted, i.e., XML elements, element contents within the target SOAP
1665 | envelope.
1666 | Encrypt the data items as follows: For each XML element or element content within the target
1667 | SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification
1668 | [XMLENC]. Each selected original element or element content MUST be removed and replaced
1669 | by the resulting <xenc:EncryptedData> element.
1670 | The optional <ds:KeyInfo> element in the <xenc:EncryptedData> element MAY reference
1671 | another <ds:KeyInfo> element. Note that if the encryption is based on an attached security
1672 | token, then a <wsse:SecurityTokenReference> element SHOULD be added to the
1673 | <ds:KeyInfo> element to facilitate locating it.
1674 | Create an <xenc:DataReference> element referencing the generated
1675 | <xenc:EncryptedData> elements. Add the created <xenc:DataReference> element to the
1676 | <xenc:ReferenceList>.
1677 | Copy all non-encrypted data.

Formatted: No bullets or numbering

1678 | 9.4.2 Decryption

1679 | On receiving a SOAP envelope containing encryption header elements, for each encryption
1680 | header element the following general steps should be processed (this section is non-normative.
1681 | Additionally, if the target of reference is an EncryptedHeader, processing rules as defined in
1682 | section 9.5.4 below SHOULD be used):

- 1683 | 1. Identify any decryption keys that are in the recipient's possession, then identifying any
1684 | message elements that it is able to decrypt.
- 1685 | 2. Locate the <xenc:EncryptedData> items to be decrypted (possibly using the
1686 | <xenc:ReferenceList>).
- 1687 | 3. Decrypt them as follows:
 - 1688 | a. For each element in the target SOAP envelope, decrypt it according to the
1689 | processing rules of the XML Encryption specification and the processing rules
1690 | listed above.
 - 1691 | b. If the decryption fails for some reason, applications MAY report the failure to the
1692 | producer using the fault code defined in Section 12 Error Handling of this
1693 | specification.
 - 1694 | c. It is possible for overlapping portions of the SOAP message to be encrypted in
1695 | such a way that they are intended to be decrypted by SOAP nodes acting in
1696 | different Roles. In this case, the <xenc:ReferenceList> or
1697 | <xenc:EncryptedKey> elements identifying these encryption operations will
1698 | necessarily appear in different <wsse:Security> headers. Since SOAP does
1699 | not provide any means of specifying the order in which different Roles will
1700 |

Deleted: 28

Deleted: June

1701 process their respective headers, this order is not specified by this specification
1702 and can only be determined by a prior agreement.

1703 9.4.3 Encryption with EncryptedHeader

1704 When it is required that an entire SOAP header block including the top-level element and its
1705 attributes be encrypted, the original header block SHOULD be replaced with a
1706 <wsse11:EncryptedHeader> element. The <wsse11:EncryptedHeader> element MUST contain
1707 the <xenc:EncryptedData> produced by encrypting the header block. A wsu:Id attribute MAY be
1708 added to the <wsse11:EncryptedHeader> element for referencing. If the referencing
1709 <wsse:Security> header block defines a value for the <S12:mustUnderstand> or
1710 <S11:mustUnderstand> attribute, that attribute and associated value MUST be copied to the
1711 <wsse11:EncryptedHeader> element. If the referencing <wsse:Security> header block defines a
1712 value for the S12:role or S11:actor attribute, that attribute and associated value MUST be copied
1713 to the <wsse11:EncryptedHeader> element. If the referencing <wsse:Security> header block
1714 defines a value for the S12:relay attribute, that attribute and associated value MUST be copied
1715 to the <wsse11:EncryptedHeader> element.

1716 Any header block can be replaced with a corresponding <wsse11:EncryptedHeader> header
1717 block. This includes <wsse:Security> header blocks. (In this case, obviously if the encryption
1718 operation is specified in the same security header or in a security header targeted at a node
1719 which is reached after the node targeted by the <wsse11:EncryptedHeader> element, the
1720 decryption will not occur.)

1721 In addition, <wsse11:EncryptedHeader> header blocks can be super-encrypted and replaced
1722 by other <wsse11:EncryptedHeader> header blocks (for wrapping/tunneling scenarios). Any
1723 <wsse:Security> header that encrypts a header block targeted to a particular actor SHOULD
1724 be targeted to that same actor, unless it is a security header.

1727 9.4.4 Processing an EncryptedHeader

1728 The processing model for <wsse11:EncryptedHeader> header blocks is as follows:

- 1729 1. Resolve references to encrypted data specified in the <wsse:Security> header block
1730 targeted at this node. For each reference, perform the following steps.
- 1731 2. If the referenced element does not have a qualified name of
1732 <wsse11:EncryptedHeader> then process as per section 9.5.2 Decryption and stop
1733 the processing steps here.
- 1734 3. Otherwise, extract the <xenc:EncryptedData> element from the
1735 <wsse11:EncryptedHeader> element.
- 1736 4. Decrypt the contents of the <xenc:EncryptedData> element as per section 9.5.2
1737 Decryption and replace the <wsse11:EncryptedHeader> element with the decrypted
1738 contents.
- 1739 5. Process the decrypted header block as per SOAP processing guidelines.

1740

Deleted: R

Deleted: A

Formatted: (Asian) Japanese

Formatted: Font: Courier New,
(Asian) Japanese

Formatted: (Asian) Japanese

Formatted: Font: Courier New,
(Asian) Japanese

Formatted: (Asian) Japanese

Formatted: Font: (Default)
Helvetica, 10 pt, (Asian) Japanese

Deleted: 28

Deleted: June

1741 Alternatively, a processor may perform a pre-pass over the encryption references in the
1742 <wsse:Security> header:

- 1743 1. Resolve references to encrypted data specified in the <wsse:Security> header block
1744 targeted at this node. For each reference, perform the following steps.
- 1745 2. If a referenced element has a qualified name of <wsse11:EncryptedHeader> then
1746 replace the <wsse11:EncryptedHeader> element with the contained
1747 <xenc:EncryptedData> element and if present copy the value of the wsu:Id attribute
1748 from the <wsse11:EncryptedHeader> element to the <xenc:EncryptedData>
1749 element.
- 1750 3. Process the <wsse:Security> header block as normal.

1751

1752 It should be noted that the results of decrypting a <wsse11:EncryptedHeader> header block
1753 could be another <wsse11:EncryptedHeader> header block. In addition, the result MAY be
1754 targeted at a different role than the role processing the <wsse11:EncryptedHeader> header
1755 block.

1756 9.4.5 Processing the mustUnderstand attribute on EncryptedHeader

1757 If the S11:mustUnderstand or S12:mustUnderstand attribute is specified on the
1758 <wsse11:EncryptedHeader> header block, and is true, then the following steps define what it
1759 means to "understand" the <wsse11:EncryptedHeader> header block:

- 1760 1. The processor MUST be aware of this element and know how to decrypt and convert into
1761 the original header block. This DOES NOT REQUIRE that the process know that it has
1762 the correct keys or support the indicated algorithms.
- 1763 2. The processor MUST, after decrypting the encrypted header block, process the
1764 decrypted header block according to the SOAP processing guidelines. The receiver
1765 MUST raise a fault if any content required to adequately process the header block
1766 remains encrypted or if the decrypted SOAP header is not understood and the value of
1767 the S12:mustUnderstand or S11:mustUnderstand attribute on the decrypted
1768 header block is true. Note that in order to comply with SOAP processing rules in this
1769 case, the processor must roll back any persistent effects of processing the security
1770 header, such as storing a received token.

1771

1772

10 Security Timestamps

1773

It is often important for the recipient to be able to determine the *freshness* of security semantics.

1774

In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.

1775

This specification does not provide a mechanism for synchronizing time. The assumption is that time is trusted or additional mechanisms, not described here, are employed to prevent replay.

1776

This specification defines and illustrates time references in terms of the `xsd:dateTime` type defined in XML Schema. It is RECOMMENDED that all time references use this type. **All**

1777

references **MUST** be in UTC time. Implementations **MUST NOT** generate time instants that

1778

specify leap seconds. If, however, other time types are used, then the `ValueType` attribute

1781

(described below) **MUST** be specified to indicate the data type of the time format. Requestors and

1782

receivers **SHOULD NOT** rely on other applications supporting time resolution finer than

1783

milliseconds.

1784

1785

The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and

1786

expiration times of the security semantics in a message.

1787

1788

All times **MUST** be in UTC format as specified by the XML Schema type (`dateTime`). It should be

1789

noted that times support time precision as defined in the XML Schema specification.

1790

The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and

1791

may only be present at most once per header (that is, per SOAP actor/role).

1792

1793

The ordering within the element is as illustrated below. The ordering of elements in the

1794

`<wsu:Timestamp>` element is fixed and **MUST** be preserved by intermediaries.

1795

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

1796

1797

```
<wsu:Timestamp wsu:Id="...">
```

1798

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

1799

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

1800

```
  ...
```

1801

```
</wsu:Timestamp>
```

1802

1803

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

1804

1805

`/wsu:Timestamp`

1806

This is the element for indicating **security semantics**, timestamps.

1807

1808

`/wsu:Timestamp/wsu:Created`

1809

This represents the creation time of the security semantics. This element is optional, but

1810

can only be specified once in a `<wsu:Timestamp>` element. Within the SOAP

1811

processing model, creation is the instant that the infonet is serialized for transmission.

1812

The creation time of the message **SHOULD NOT** differ substantially from its transmission

1813

time. The difference in time should be minimized.

1814

1815

`/wsu:Timestamp/wsu:Expires`

WSS: SOAP Message Security (WS-Security 2004)

Copyright © OASIS Open 2002-2005. All Rights Reserved.

11 October 2005

Page 50 of 74

Deleted: It is further RECOMMENDED that all

Deleted: message

Deleted: 28

Deleted: June

1816 This element represents the expiration of the security semantics. This is optional, but
1817 can appear at most once in a `<wsu:Timestamp>` element. Upon expiration, the
1818 requestor asserts that its security semantics are no longer valid. It is strongly
1819 RECOMMENDED that recipients (anyone who processes this message) discard (ignore)
1820 any message whose security semantics have passed their expiration. A Fault code
1821 (`wsu:MessageExpired`) is provided if the recipient wants to inform the requestor that its
1822 security semantics were expired. A service MAY issue a Fault indicating the security
1823 semantics have expired.
1824

1825 `/wsu:Timestamp/{any}`
1826 This is an extensibility mechanism to allow additional elements to be added to the
1827 element. Unrecognized elements SHOULD cause a fault.
1828

1829 `/wsu:Timestamp/@wsu:Id`
1830 This optional attribute specifies an XML Schema ID that can be used to reference this
1831 element (the timestamp). This is used, for example, to reference the timestamp in a XML
1832 Signature.
1833

1834 `/wsu:Timestamp/@{any}`
1835 This is an extensibility mechanism to allow additional attributes to be added to the
1836 element. Unrecognized attributes SHOULD cause a fault.
1837

1838 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
1839 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
1840 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
1841 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
1842 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
1843 judgment of the requestor's likely current clock time by means not described in this specification,
1844 for example an out-of-band clock synchronization protocol. The recipient may also use the
1845 creation time and the delays introduced by intermediate SOAP roles to estimate the degree of
1846 clock skew.
1847

1848 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```
1849 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">  
1851 <S11:Header>  
1852 <wsse:Security>  
1853 <wsu:Timestamp wsu:Id="timestamp">  
1854 <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>  
1855 <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>  
1856 </wsu:Timestamp>  
1857 ...  
1858 </wsse:Security>  
1859 ...  
1860 </S11:Header>  
1861 <S11:Body>  
1862 ...  
1863 </S11:Body>  
1864 </S11:Envelope>
```

1865

11 Extended Example

1866

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

1867

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

1868

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

1869

<wsse:Security> header.

1870

1871

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
(003)   xmlns:xenc="..." xmlns:ds="...">
(004)   <S11:Header>
(005)     <wsse:Security>
(006)       <wsu:Timestamp wsu:Id="T0">
(007)         <wsu:Created>
(008)           2001-09-13T08:42:00Z</wsu:Created>
(009)         </wsu:Timestamp>
(010)       <wsse:BinarySecurityToken
(011)         ValueType="...#X509v3"
(012)         wsu:Id="X509Token"
(013)         EncodingType="...#Base64Binary">
(014)         MIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
(015)       </wsse:BinarySecurityToken>
(016)       <xenc:EncryptedKey>
(017)         <xenc:EncryptionMethod Algorithm=
(018)           "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
(019)         <ds:KeyInfo>
(020)           <wsse:SecurityTokenReference>
(021)             <wsse:KeyIdentifier
(022)               EncodingType="...#Base64Binary"
(023)               ValueType="...#X509v3">MIGfMa0GCSq...
(024)             </wsse:KeyIdentifier>
(025)           </ds:KeyInfo>
(026)         <xenc:CipherData>
(027)           <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(028)         </xenc:CipherValue>
(029)       </xenc:EncryptedKey>
(030)       <ds:Signature>
(031)         <ds:SignedInfo>
(032)           <ds:CanonicalizationMethod
(033)             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(034)           <ds:SignatureMethod
(035)             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(036)           <ds:Reference URI="#T0">
(037)             <ds:Transforms>
(038)               <ds:Transform
(039)                 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
```

1914

Deleted: 28

Deleted: June

```

1915 | (034)         </ds:Transforms>
1916 | (035)         <ds:DigestMethod
1917 |               Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1918 | (036)         <ds:DigestValue>LyLsF094hPi4wPU...
1919 | (037)         </ds:DigestValue>
1920 | (038)         </ds:Reference>
1921 | (039)         <ds:Reference URI="#body">
1922 | (040)         <ds:Transforms>
1923 | (041)         <ds:Transform
1924 |               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1925 | (042)         </ds:Transforms>
1926 | (043)         <ds:DigestMethod
1927 |               Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1928 | (044)         <ds:DigestValue>LyLsF094hPi4wPU...
1929 | (045)         </ds:DigestValue>
1930 | (046)         </ds:Reference>
1931 | (047)         </ds:SignedInfo>
1932 | (048)         <ds:SignatureValue>
1933 | (049)         HplZkmFZ/2kQLXDJbchm5gK...
1934 | (050)         </ds:SignatureValue>
1935 | (051)         <ds:KeyInfo>
1936 | (052)         <wsse:SecurityTokenReference>
1937 | (053)         <wsse:Reference URI="#X509Token" />
1938 | (054)         </wsse:SecurityTokenReference>
1939 | (055)         </ds:KeyInfo>
1940 | (056)         </ds:Signature>
1941 | (057)         </wsse:Security>
1942 | (058)         </S11:Header>
1943 | (059)         <S11:Body wsu:Id="body">
1944 | (060)         <xenc:EncryptedData
1945 |               Type="http://www.w3.org/2001/04/xmlenc#Element"
1946 |               wsu:Id="encl1">
1947 | (061)         <xenc:EncryptionMethod
1948 |               Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-
1949 |               cbc" />
1950 | (062)         <xenc:CipherData>
1951 | (063)         <xenc:CipherValue>d2FpbmdvbGRFE0lm4byV0...
1952 | (064)         </xenc:CipherValue>
1953 | (065)         </xenc:CipherData>
1954 | (066)         </xenc:EncryptedData>
1955 | (067)         </S11:Body>
1956 | (068) </S11:Envelope>

```

1957
1958 | Let's review some of the key sections of this example:
1959 | Lines (003)-(058) contain the SOAP message headers.

1960
1961 | Lines (004)-(057) represent the <wsse:Security> header block. This contains the security-
1962 | related information for the message.

1963
1964 | Lines (005)-(008) specify the timestamp information. In this case it indicates the creation time of
1965 | the security semantics.

Deleted: 28

Deleted: June

1967 | Lines (010)-(012) specify a security token that is associated with the message. In this case, it
1968 | specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64
1969 | encoding of the certificate.
1970 |
1971 | Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a
1972 | symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
1973 | encrypt the key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the
1974 | symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines
1975 | (023)-(025) identify the encryption block in the message that uses this symmetric key. In this
1976 | case it is only used to encrypt the body (Id="enc1").
1977 |
1978 | Lines (027)-(056) specify the digital signature. In this example, the signature is based on the
1979 | X.509 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039)
1980 | references the message body.
1981 |
1982 | Lines (048)-(050) indicate the actual signature value – specified in Line (043).
1983 |
1984 | Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the X.509
1985 | certificate included in the message. Line (053) provides a URI link to the Lines (010)-(012).
1986 | The body of the message is represented by Lines (059)-(067).
1987 |
1988 | Lines (060)-(066) represent the encrypted metadata and form of the body using XML Encryption.
1989 | Line (060) indicates that the "element value" is being replaced and identifies this encryption. Line
1990 | (061) specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the
1991 | actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
1992 | key as the key references this encryption – Line (024).
1993 |

1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016

12 Error Handling

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

- Invalid or unsupported type of security token, signing, or encryption
- Invalid or unauthenticated or unauthenticatable security token
- Invalid signature
- Decryption failure
- Referenced security token is unavailable
- Unsupported namespace

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.25" + Indent at: 0.25"

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.

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.

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

Deleted: F

2017
2018
2019

The "failure" class of errors are:

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

Deleted: 28

Deleted: June

Referenced security token could not be retrieved	wsse:SecurityTokenUnavailable
<u>The message has expired</u>	<u>wsse:MessageExpired</u>

Deleted: 28
Deleted: June

2020

13 Security Considerations

2021

2022

2023

2024

2025

2026

2027

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.

2028

13.1 General Considerations

2029

2030

2031

2032

2033

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*.

2034

2035

2036

2037

2038

2039

2040

2041

2042

2043

2044

2045

2046

2047

2048

- 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 utmost 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)

2049

2050

2051

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.

2052

13.2 Additional Considerations

2053

13.2.1 Replay

2054

2055

2056

2057

Digital signatures alone do not provide message authentication. One can record a signed message and resend it (a replay attack). It is strongly **RECOMMENDED** that messages include digitally signed elements to allow message recipients to detect replays of the message when the messages are exchanged via an open network. These can be part of the message or of the

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

11, October 2005
Page 57 of 74

Deleted: 28

Deleted: June

2058 headers defined from other SOAP extensions. Four typical approaches are: Timestamp,
2059 Sequence Number, Expirations and Message Correlation. Signed timestamps MAY be used to
2060 keep track of messages (possibly by caching the most recent timestamp from a specific service)
2061 and detect replays of previous messages. It is RECOMMENDED that timestamps be cached for
2062 a given period of time, as a guideline, a value of five minutes can be used as a minimum to detect
2063 replays, and that timestamps older than that given period of time set be rejected in interactive
2064 scenarios.

2065 **13.2.2 Combining Security Mechanisms**

2066 This specification defines the use of XML Signature and XML Encryption in SOAP headers. As
2067 one of the building blocks for securing SOAP messages, it is intended to be used in conjunction
2068 with other security techniques. Digital signatures need to be understood in the context of other
2069 security mechanisms and possible threats to an entity.

2070 Implementers should also be aware of all the security implications resulting from the use of digital
2071 signatures in general and XML Signature in particular. When building trust into an application
2072 based on a digital signature there are other technologies, such as certificate evaluation, that must
2073 be incorporated, but these are outside the scope of this document.

2074
2075
2076 As described in XML Encryption, the combination of signing and encryption over a common data
2077 item may introduce some cryptographic vulnerability. For example, encrypting digitally signed
2078 data, while leaving the digital signature in the clear, may allow plain text guessing attacks.

2079 **13.2.3 Challenges**

2080 When digital signatures are used for verifying the claims pertaining to the sending entity, the
2081 producer must demonstrate knowledge of the confirmation key. One way to achieve this is to use
2082 a challenge-response type of protocol. Such a protocol is outside the scope of this document.
2083 To this end, the developers can attach timestamps, expirations, and sequences to messages.

2084 **13.2.4 Protecting Security Tokens and Keys**

2085 Implementers should be aware of the possibility of a token substitution attack. In any situation
2086 where a digital signature is verified by reference to a token provided in the message, which
2087 specifies the key, it may be possible for an unscrupulous producer to later claim that a different
2088 token, containing the same key, but different information was intended.
2089 An example of this would be a user who had multiple X.509 certificates issued relating to the
2090 same key pair but with different attributes, constraints or reliance limits. Note that the signature of
2091 the token by its issuing authority does not prevent this attack. Nor can an authority effectively
2092 prevent a different authority from issuing a token over the same key if the user can prove
2093 possession of the secret.

2094
2095 The most straightforward counter to this attack is to insist that the token (or its unique identifying
2096 data) be included under the signature of the producer. If the nature of the application is such that
2097 the contents of the token are irrelevant, assuming it has been issued by a trusted authority, this
2098 attack may be ignored. However because application semantics may change over time, best
2099 practice is to prevent this attack.
2100

2101 Requestors should use digital signatures to sign security tokens that do not include signatures (or
2102 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
2103 RECOMMENDED that all relevant and immutable message content be signed by the producer.
2104 Receivers SHOULD only consider those portions of the document that are covered by the
2105 producer's signature as being subject to the security tokens in the message. Security tokens
2106 appearing in <wsse:Security> header elements SHOULD be signed by their issuing authority
2107 so that message receivers can have confidence that the security tokens have not been forged or
2108 altered since their issuance. It is strongly RECOMMENDED that a message producer sign any
2109 <wsse:SecurityToken> elements that it is confirming and that are not signed by their issuing
2110 authority.
2111 When a requester provides, within the request, a Public Key to be used to encrypt the response,
2112 it is possible that an attacker in the middle may substitute a different Public Key, thus allowing the
2113 attacker to read the response. The best way to prevent this attack is to bind the encryption key in
2114 some way to the request. One simple way of doing this is to use the same key pair to sign the
2115 request as to encrypt the response. However, if policy requires the use of distinct key pairs for
2116 signing and encryption, then the Public Key provided in the request should be included under the
2117 signature of the request.

2118 13.2.5 Protecting Timestamps and Ids

2119 In order to *trust* wsu:Id attributes and <wsu:Timestamp> elements, they SHOULD be signed
2120 using the mechanisms outlined in this specification. This allows readers of the IDs and
2121 timestamps information to be certain that the IDs and timestamps haven't been forged or altered
2122 in any way. It is strongly RECOMMENDED that IDs and timestamp elements be signed.
2123

2124 13.2.6 Protecting against removal and modification of XML Elements

2125 XML Signatures using Shorthand XPointer References (AKA IDREF) protect against the removal
2126 and modification of XML elements; but do not protect the location of the element within the XML
2127 Document.

2128 Whether or not this is a security vulnerability depends on whether the location of the signed data
2129 within its surrounding context has any semantic import. This consideration applies to data carried
2130 in the SOAP Body or the Header.

2131 Of particular concern is the ability to relocate signed data into a SOAP Header block which is
2132 unknown to the receiver and marked mustUnderstand="false". This could have the effect of
2133 causing the receiver to ignore signed data which the sender expected would either be processed
2134 or result in the generation of a MustUnderstand fault.

2135 A similar exploit would involve relocating signed data into a SOAP Header block targeted to a
2136 S11:actor or S12:role other than that which the sender intended, and which the receiver will not
2137 process.

2138 While these attacks could apply to any portion of the message, their effects are most pernicious
2139 with SOAP header elements which may not always be present, but must be processed whenever
2140 they appear.

2141
2142
2143
2144
2145

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

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

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

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

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

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

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

Deleted: 28

Deleted: June

... [2]

2146 In the general case of XML Documents and Signatures, this issue may be resolved by signing the
2147 entire XML Document and/or strict XML Schema specification and enforcement. However,
2148 because elements of the SOAP message, particularly header elements, may be legitimately
2149 modified by SOAP intermediaries, this approach is usually not appropriate. It is RECOMMENDED
2150 that applications signing any part of the SOAP body sign the entire body.

2151 Alternatives countermeasures include (but are not limited to):

- 2153 • References using XPath transforms with Absolute Path expressions with checks
2154 performed by the receiver that the URI and Absolute Path XPath expression evaluate to
2155 the digested nodeset.
- 2156 • A Reference using an XPath transform to include any significant location-dependent
2157 elements and exclude any elements that might legitimately be removed, added, or altered
2158 by intermediaries.
- 2159 • Using only References to elements with location-independent semantics.
- 2160 • Strict policy specification and enforcement regarding which message parts are to be
2161 signed. For example:
 - 2162 ○ Requiring that the entire SOAP Body and all children of SOAP Header be signed.
 - 2163 ○ Requiring that SOAP header elements which are marked
2164 MustUnderstand="false" and have signed descendants MUST include the
2165 MustUnderstand attribute under the signature.

Formatted: Indent: Left: 0"

Deleted: <#>¶

Formatted: Bullets and Numbering

2167 13.2.7 Detecting Duplicate Identifiers

2168 The wsse:Security processing SHOULD check for duplicate values from among the set of ID
2169 attributes that it is aware of. The wsse:Security processing MUST generate a fault if a duplicate
2170 ID value is detected.

2171 This section is non-normative.

Deleted: ¶

Deleted: 28

Deleted: June

2173

14 Interoperability Notes

2174

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.

2175

2176

2177

2178

2179

2180

2181

2182

2183

2184

2185

2186

2187

2188

2189

2190

2191

2192

2193

2194

2195

2196

2197

2198

2199

2200

- **Key Identifiers:** Make sure you understand the algorithm and how it is applied to security tokens.
- **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.
- **Encryption Padding:** The XML Encryption random block cipher padding has caused issues with certain decryption implementations; be careful to follow the specifications exactly.
- **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.
- **Time Formats:** This specification uses a restricted version of the XML Schema `xsd:dateTime` element. Take care to ensure compliance with the specified restrictions.
- **Byte Order Marker (BOM):** Some implementations have problems processing the BOM marker. It is suggested that usage of this be optional.
- **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.

This section is non-normative.

Formatted: Font: Courier New

Deleted: d

Formatted: Indent: Left: 0"

Deleted: 28

Deleted: June

2201

15 Privacy Considerations

2202

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.

2203

2204

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

2205

2206

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

2207

2208

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

2209

2210

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

2211

2212

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

2213

2214

typically require that the actors are referenced in an identifiable way; such identifiable references

2215

2216

are also typically needed to obtain appropriate encryption keys for the intermediaries. If intermediaries add claims, they should be guided by their privacy policies just like the original producers.

2217

2218

2219

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

2220

2221

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

2222

2223

This section is non-normative.

Deleted: 28

Deleted: June

16References

2224

- 2225 **[GLOSS]** Informational RFC 2828, "Internet Security Glossary," May 2000.
- 2226 **[KERBEROS]** J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993, <http://www.ietf.org/rfc/rfc1510.txt> .
- 2227
- 2228 **[KEYWORDS]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997
- 2229
- 2230 **[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>
- 2231
- 2232
- 2233 **[SOAP11]** W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.
- 2234 **[SOAP12]** W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework", 23 June 2003
- 2235
- 2236 **[SOAPSEC]** W3C Note, "SOAP Security Extensions: Digital Signature," 06 February 2001.
- 2237
- 2238 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 3986, MIT/LCS, Day Software, Adobe Systems, January 2005.
- 2239
- 2240
- 2241 **[XPATH]** W3C Recommendation, "XML Path Language", 16 November 1999
- 2242

2243 The following are non-normative references included for background and related material:

- 2244 **[WS-SECURITY]** "Web Services Security Language", IBM, Microsoft, VeriSign, April 2002.
- 2245 "WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002.
- 2246 "WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002.
- 2247 **[XMLC14N]** W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001
- 2248 **[EXCC14N]** W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8 July 2002.
- 2249
- 2250 **[XMLENC]** W3C Working Draft, "XML Encryption Syntax and Processing," 04 March 2002
- 2251
- 2252 W3C Recommendation, "Decryption Transform for XML Signature", 10 December 2002. Formatted: Indent: Hanging: 1.25"
- 2253 **[XML-ns]** W3C Recommendation, "Namespaces in XML," 14 January 1999.
- 2254 **[XMLSCHEMA]** W3C Recommendation, "XML Schema Part 1: Structures," 2 May 2001.
- 2255 W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.
- 2256 **[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. <http://www.w3.org/TR/xmlsig-core/>.
- 2257
- 2258

2259 2260 2261 2262	[X509]	S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified Certificates Profile," http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-l
2263 2264	[WSS-SAML]	OASIS Working Draft 06, "Web Services Security SAML Token Profile", 21 February 2003
2265 2266	[WSS-XrML]	OASIS Working Draft 03, "Web Services Security XrML Token Profile", 30 January 2003
2267 2268 2269	[WSS-X509]	OASIS, "Web Services Security X.509 Certificate Token Profile", 19 January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0
2270 2271	[WSSKERBEROS]	OASIS Working Draft 03, "Web Services Security Kerberos Profile", 30 January 2003
2272 2273 2274	[WSSUSERNAME]	OASIS, "Web Services Security UsernameToken Profile" 19 January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0
2275 2276	[WSS-XCBF]	OASIS Working Draft 1.1, "Web Services Security XCBF Token Profile", 30 March 2003
2277 2278	[XPOINTER]	"XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation", DeRose, Maler, Daniel, 11 September 2001.

Deleted: 28

Deleted: June

2279

Appendix A: Acknowledgements

2280

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	Fujiitsu
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 2002-2005. All Rights Reserved.

11 October 2005
Page 65 of 74

- Deleted: Gene ... [3]
- Formatted ... [4]
- Formatted Table ... [5]
- Formatted ... [6]
- Formatted ... [7]
- Formatted ... [8]
- Formatted ... [9]
- Formatted ... [10]
- Formatted ... [11]
- Formatted ... [12]
- Formatted ... [13]
- Formatted ... [14]
- Formatted ... [15]
- Formatted ... [16]
- Formatted ... [17]
- Formatted ... [18]
- Formatted ... [19]
- Formatted ... [20]
- Formatted ... [21]
- Formatted ... [22]
- Formatted ... [23]
- Formatted ... [24]
- Formatted ... [25]
- Formatted ... [26]
- Formatted ... [27]
- Formatted ... [28]
- Formatted ... [29]
- Formatted ... [30]
- Formatted ... [31]
- Formatted ... [32]
- Formatted ... [33]
- Formatted ... [34]
- Formatted ... [35]
- Formatted ... [36]
- Formatted ... [37]
- Formatted ... [38]
- Formatted ... [39]
- Formatted ... [40]
- Formatted ... [41]
- Formatted ... [42]
- Formatted ... [43]
- Formatted ... [44]
- Formatted ... [45]
- Deleted: 28...June ... [46]

2281

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

Previous Contributors:

Pete	Dapkus	BEA
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard
Xin	Wang	ContentGuard
Shawn	Sharp	Cyclone Commerce
Ganesh	Vaideeswaran	Documentum
Tim	Moses	Entrust
Carolina	Canales-Valenzuela	Ericsson
Tom	Rutt	Fujitsu
Yutaka	Kudo	Hitachi
Jason	Rouault	HP
Bob	Blakley	IBM
Joel	Farrell	IBM
Satoshi	Hada	IBM
Hiroshi	Maruyama	IBM
David	Melgar	IBM
Kent	Tamura	IBM
Wayne	Vicknair	IBM
Phil	Griffin	Individual
Mark	Hayes	Individual
John	Hughes	Individual
Peter	Rostin	Individual
Davanum	Srinivas	Individual
Bob	Morgan	Individual/Internet
Bob	Atkinson	Microsof
Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Giovanni	Della-Libera	Microsoft
Alan	Geller	Microsoft
Johannes	Klein	Microsoft
Scott	Konersmann	Microsoft

- Formatted ... [47]
- Formatted ... [48]
- Formatted ... [49]
- Formatted ... [50]
- Formatted ... [51]
- Formatted ... [52]
- Formatted ... [53]
- Formatted ... [54]
- Formatted ... [55]
- Formatted ... [56]
- Formatted ... [57]
- Formatted ... [58]
- Formatted ... [59]
- Formatted ... [60]
- Formatted Table ... [61]
- Formatted ... [62]
- Formatted ... [63]
- Formatted ... [64]
- Formatted ... [65]
- Formatted ... [66]
- Formatted ... [67]
- Formatted ... [68]
- Formatted ... [69]
- Formatted ... [70]
- Formatted ... [71]
- Formatted ... [72]
- Formatted ... [73]
- Formatted ... [74]
- Formatted ... [75]
- Formatted ... [76]
- Formatted ... [77]
- Formatted ... [78]
- Formatted ... [79]
- Formatted ... [80]
- Formatted ... [81]
- Formatted ... [82]
- Formatted ... [83]
- Formatted ... [84]
- Formatted ... [85]
- Formatted ... [86]
- Formatted ... [87]
- Formatted ... [88]
- Formatted ... [89]
- Formatted ... [90]
- Formatted ... [91]
- Deleted: 28...June ... [92]

2283

Appendix B: Revision History

Rev	Date	By Whom	What
WGD 1.1	2005-07-24	Anthony Nadalin	Issue 310, 334, 389, 403
WGD 1.1	2005-08-30	Anthony Nadalin	Issue 411
WGD 1.1	2005-10-11	Anthony Nadalin	Issue 334, 405, 432, 433, 436, 439, 443, 445

Deleted: 2004-09-13

Deleted: Initial version cloned from the Version 1.1 and Errata

Deleted: WGD 1.1

... [93]

2284

2285

This section is non-normative.

Deleted: 28

Deleted: June

Appendix C: Utility Elements and Attributes

2287 These specifications define several elements, attributes, and attribute groups which can be re-
 2288 used by other specifications. This appendix provides an overview of these *utility* components. It
 2289 should be noted that the detailed descriptions are provided in the specification and this appendix
 2290 will reference these sections as well as calling out other aspects not documented in the
 2291 specification.

2292 16.1 Identification Attribute

2293 There are many situations where elements within SOAP messages need to be referenced. For
 2294 example, when signing a SOAP message, selected elements are included in the signature. XML
 2295 Schema Part 2 provides several built-in data types that may be used for identifying and
 2296 referencing elements, but their use requires that consumers of the SOAP message either have or
 2297 are able to obtain the schemas where the identity or reference mechanisms are defined. In some
 2298 circumstances, for example, intermediaries, this can be problematic and not desirable.

2299 Consequently a mechanism is required for identifying and referencing elements, based on the
 2300 SOAP foundation, which does not rely upon complete schema knowledge of the context in which
 2301 an element is used. This functionality can be integrated into SOAP processors so that elements
 2302 can be identified and referred to without dynamic schema discovery and processing.

2303 This specification specifies a namespace-qualified global attribute for identifying an element
 2304 which can be applied to any element that either allows arbitrary attributes or specifically allows
 2305 this attribute. This is a general purpose mechanism which can be re-used as needed.
 2306 A detailed description can be found in Section 4.0 ID References.

2307 This section is non-normative.
 2308
 2309
 2310

2311 16.2 Timestamp Elements

2312 The specification defines XML elements which may be used to express timestamp information
 2313 such as creation and expiration. While defined in the context of message security, these
 2314 elements can be re-used wherever these sorts of time statements need to be made.

2315 The elements in this specification are defined and illustrated using time references in terms of the
 2316 *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this
 2317 type for interoperability. It is further RECOMMENDED that all references be in UTC time for
 2318 increased interoperability. If, however, other time types are used, then the `valueType` attribute
 2319 MUST be specified to indicate the data type of the time format.

2320 The following table provides an overview of these elements:
 2321
 2322

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.

2323
2324
2325
2326
2327

A detailed description can be found in Section 10.
This section is non-normative.

16.3 General Schema Types

2328
2329
2330
2331
2332
2333
2334
2335

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

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

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

2336
2337
2338

This section is non-normative.

Deleted: 28
Deleted: June

2339

Appendix D: SecurityTokenReference Model

2340

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

2341

2342

2343

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

2344

2345

- 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.

2346

2347

2348

2349

2350

2351

2352

2353

2354

2355

The following use cases drive the above motivations:

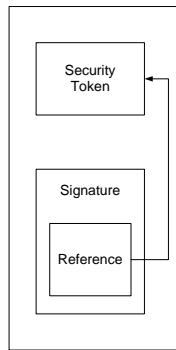
2356

2357

2358

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

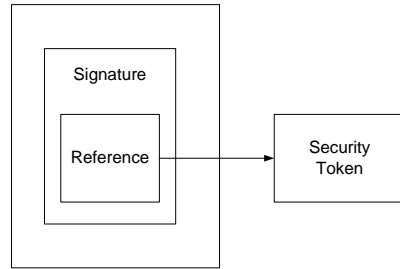
2359



2360

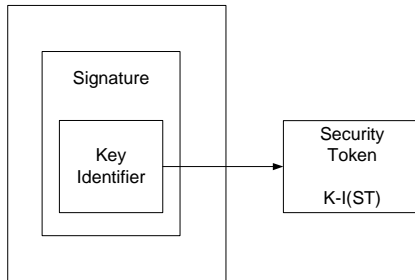
2361
2362
2363
2364

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:



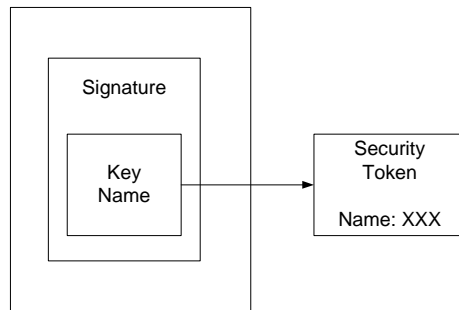
2365
2366
2367
2368

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:



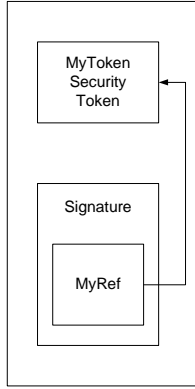
2369
2370
2371
2372

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:



2373
2374
2375

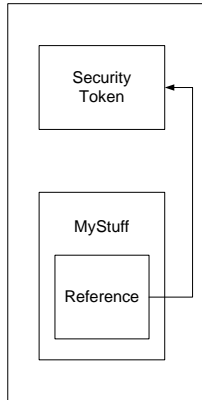
Format-Specific References – A security token is associated with an XML Signature and identified using a mechanism specific to the token (rather than the general mechanisms



2376 described above). The figure below illustrates this:

2377

2378 **Non-Signature References** – A message may contain XML that does not represent an XML



2379 signature, but may reference a security token (which may or may not be included in the
2380 message). The figure below illustrates this:

2381

2382

2383 | All conformant implementations **must** be able to process the
2384 `<wss:SecurityTokenReference>` element. However, they are not required to support all of
2385 the different types of references.

Deleted: MUST

2386

2387 | The reference **may** include a `wss11:TokenType` attribute which provides a "hint" for the type of
2388 desired token.

Deleted: MAY

2389

2390 If multiple sub-elements are specified, together they describe the reference for the token.

Deleted: ValueType

2391 There are several challenges that implementations face when trying to interoperate:

Formatted: Font: Not Italic

2392 **ID References** – The underlying XML referencing mechanism using the XML base type of ID
2393 provides a simple straightforward XML element reference. However, because this is an XML
2394 type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
2395 requires the recipient to *understand* the schema. This may be an expensive task and in the
2396 general case impossible as there is no way to know the "schema location" for a specific
2397 namespace URI.

Formatted: Font: Not Italic

Formatted: Font: Not Italic

2398

Deleted: 28

Deleted: June

2399 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
2400 references are, by definition, unique by XML. However, other mechanisms such as "principal
2401 name" are not required to be unique and therefore such references may be unique.
2402 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
2403 information about the "key" used in the signature. For token references within signatures, it is
2404 recommended that the `<wsse:SecurityTokenReference>` be placed within the
2405 `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
2406 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
2407 Message Security or its profiles are preferred over the mechanisms in XML Signature.
2408 The following provides additional details on the specific reference mechanisms defined in WSS:
2409 SOAP Message Security:

Deleted: RECOMMENDED

2410
2411 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
2412 the security token. If only the fragment is specified, then it references the security token within
2413 the document whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to
2414 a [potentially external] security token identified using a URI. There are no implied semantics
2415 around the processing of the URI.

2416
2417 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
2418 by specifying a known value (identifier) for the token, which is determined by applying a special
2419 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
2420 specific security token but requires a profile or token-specific function to be specified. The
2421 `ValueType` attribute defines the type of key identifier and, consequently, identifies the type of
2422 token referenced. The `EncodingType` attribute specifies how the unique value (identifier) is
2423 encoded. For example, a hash value may be encoded using base 64 encoding.

2424
2425 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
2426 specific value that is used to *match* an identity assertion within the security token. This is a
2427 subset match and may result in multiple security tokens that match the specified name. While
2428 XML Signature doesn't imply formatting semantics, WSS: SOAP Message Security recommends
2429 that X.509 names be specified.

Deleted: RECOMMENDS

2430
2431 It is expected that, where appropriate, profiles define if and how the reference mechanisms map
2432 to the specific token profile. Specifically, the profile should answer the following questions:

- 2433
- 2434 • What types of references can be used?
 - 2435 • How "Key Name" references map (if at all)?
 - 2436 • How "Key Identifier" references map (if at all)?
 - 2437 • Are there any additional profile or format-specific references?
- 2438

2439 This section is non-normative.

Deleted: 28

Deleted: June

1	Introduction.....	7
1.1	Goals and Requirements	7
1.1.1	Requirements	8
1.1.2	Non-Goals	8
2	Notations and Terminology.....	9
2.1	Notational Conventions.....	9
2.2	Namespaces.....	9
2.3	Acronyms and Abbreviations	10
2.4	Terminology.....	10
2.5	Note on Examples	12
3	Message Protection Mechanisms.....	13
3.1	Message Security Model.....	13
3.2	Message Protection	13
3.3	Invalid or Missing Claims	14
3.4	Example.....	14
4	ID References	16
4.1	Id Attribute	16
4.2	Id Schema	16
5	Security Header.....	18
6	Security Tokens.....	21
6.1	Attaching Security Tokens	21
6.1.1	Processing Rules.....	21
6.1.2	Subject Confirmation	21
6.2	User Name Token.....	21
6.2.1	Usernames	21
6.3	Binary Security Tokens.....	22
6.3.1	Attaching Security Tokens	22
6.3.2	Encoding Binary Security Tokens	22
6.4	XML Tokens.....	23
6.5	EncryptedData Token	24
6.6	Identifying and Referencing Security Tokens	24
7	Token References	25
7.1	SecurityTokenReference Element	25
7.2	Direct References	27
7.3	Key Identifiers	28
7.4	Embedded References	30
7.5	ds:KeyInfo.....	31
7.6	Key Names	31
7.7	Encrypted Key reference	31
8	Signatures	33
8.1	Algorithms.....	33
8.2	Signing Messages	36
8.3	Signing Tokens	36

8.4	Signature Validation	39
8.5	Signature Confirmation	39
8.5.1	Response Generation Rules	40
8.5.2	Response Processing Rules	41
8.6	Example	41
9	Encryption	43
9.1	xenc:ReferenceList	43
9.2	xenc:EncryptedKey	44
9.3	Encrypted Header	45
9.4	Processing Rules	45
9.4.1	Encryption	45
9.4.2	Decryption	46
9.4.3	Encryption with EncryptedHeader	47
9.4.4	Processing an EncryptedHeader	47
9.4.5	Processing the mustUnderstand attribute on EncryptedHeader	48
10	Security Timestamps	49
11	Extended Example	51
12	Error Handling	54
13	Security Considerations	56
13.1	General Considerations	56
13.2	Additional Considerations	56
13.2.1	Replay	56
13.2.2	Combining Security Mechanisms	57
13.2.3	Challenges	57
13.2.4	Protecting Security Tokens and Keys	57
13.2.5	Protecting Timestamps and Ids	58
13.2.6	Protecting against removal and modification of XML Elements	58
13.2.7	Detecting Duplicate Identifiers	59
14	Interoperability Notes	60
15	Privacy Considerations	61
16	References	62
	Appendix A: Acknowledgements	64
	Appendix B: Revision History	67
	Appendix C: Utility Elements and Attributes	68
16.1	Identification Attribute	68
16.2	Timestamp Elements	68
16.3	General Schema Types	69
	Appendix D: SecurityTokenReference Model	70
1	Introduction	7
1.1	Goals and Requirements	7
1.1.1	Requirements	7
1.1.2	Non-Goals	8
2	Notations and Terminology	9
2.1	Notational Conventions	9

2.2	Namespaces	9
2.3	Acronyms and Abbreviations	10
2.4	Terminology	10
2.5	Note on Examples	12
3	Message Protection Mechanisms	13
3.1	Message Security Model	13
3.2	Message Protection	13
3.3	Invalid or Missing Claims	14
3.4	Example	14
4	ID References	16
4.1	Id Attribute	16
4.2	Id Schema	16
5	Security Header	18
6	Security Tokens	20
6.1	Attaching Security Tokens	20
6.1.1	Processing Rules	20
6.1.2	Subject Confirmation	20
6.2	User Name Token	20
6.2.1	Usernames	20
6.3	Binary Security Tokens	21
6.3.1	Attaching Security Tokens	21
6.3.2	Encoding Binary Security Tokens	21
6.4	XML Tokens	22
6.5	EncryptedData Token	22
6.6	Identifying and Referencing Security Tokens	23
7	Token References	24
7.1	SecurityTokenReference Element	24
7.2	Direct References	25
7.3	Key Identifiers	26
7.4	Embedded References	28
7.5	ds:KeyInfo	29
7.6	Key Names	29
7.7	Encrypted Key reference	29
8	Signatures	31
8.1	Algorithms	31
8.2	Signing Messages	33
8.3	Signing Tokens	34
8.4	Signature Validation	36
8.5	Signature Confirmation	37
8.5.1	Response Generation Rules	38
8.5.2	Response Processing Rules	38
8.6	Example	39
9	Encryption	40
9.1	xenc:ReferenceList	40

9.2 xenc:EncryptedKey	41
9.3 Encrypted Header	42
9.4 Processing Rules	42
9.4.1 Encryption	42
9.4.2 Decryption	43
9.4.3 Encryption with EncryptedHeader	43
9.4.4 Processing an EncryptedHeader	44
9.4.5 Processing the mustUnderstand attribute on EncryptedHeader	45
10 Security Timestamps	46
11 Extended Example	48
12 Error Handling	51
13 Security Considerations	52
13.1 General Considerations	52
13.2 Additional Considerations	52
13.2.1 Replay	52
13.2.2 Combining Security Mechanisms	53
13.2.3 Challenges	53
13.2.4 Protecting Security Tokens and Keys	53
13.2.5 Protecting Timestamps and Ids	54
13.2.6 Protecting against removal and modification of XML Elements	54
14 Interoperability Notes	56
15 Privacy Considerations	57
16 References	58
Appendix A: Acknowledgements	60
Appendix B: Revision History	62
Appendix C: Utility Elements and Attributes	63
16.1 Identification Attribute	63
16.2 Timestamp Elements	63
16.3 General Schema Types	64
Appendix D: SecurityTokenReference Model	65

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

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

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

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

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

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

Alternatives countermeasures include (but are not limited to):

References using XPath transforms with Absolute Path expressions,

A Reference using an XPath transform to include any significant location-dependent elements and exclude any elements that might legitimately be removed, added, or altered by intermediaries,

Using only References to elements with location-independent semantics,

Strict policy specification and enforcement regarding which message parts are to be signed. For example:

Requiring that the entire SOAP Body and all children of SOAP Header be signed,

Requiring that SOAP header elements which are marked `mustUnderstand="false"` and have signed descendents MUST include the `mustUnderstand` attribute under the signature.

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

Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Paul	Cotton	Microsoft
Giovanni	Della-Libera	Microsoft
Vijay	Gajjala	Microsoft
Johannes	Klein	Microsoft
Scott	Konersmann	Microsoft
Chris	Kurt	Microsoft
Brian	LaMacchia	Microsoft
Paul	Leach	Microsoft
John	Manferdelli	Microsoft
John	Shewchuk	Microsoft
Dan	Simon	Microsoft
Hervey	Wilson	Microsoft
Chris	Kaler	Microsoft (co-Chair)
Prateek	Mishra	Netegrity
Frederick	Hirsch	Nokia
Senthil	Sengodan	Nokia
Lloyd	Burch	Novell
Ed	Reed	Novell
Charles	Knouse	Oblix
Vipin	Samar	Oracle
Jerry	Schwarz	Oracle
Eric	Gravengaard	Reactivity
Stuart	King	Reed Elsevier
Andrew	Nash	RSA Security
Rob	Philpott	RSA Security
Peter	Rostin	RSA Security
Martijn	de Boer	SAP
Blake	Dournaee	Sarvega
Pete	Wenzel	SeeBeyond
Jonathan	Tourzan	Sony
Yassir	Elley	Sun Microsystems
Jeff	Hodges	Sun Microsystems
Ronald	Monzillo	Sun Microsystems

Jan	Alexander	Systinet
Michael	Nguyen	The IDA of Singapore
Don	Adams	TIBCO
Symon	Chang	TIBCO
John	Weiland	US Navy
Phillip	Hallam-Baker	VeriSign
Mark	Hays	Verisign
Hemma	Prafullchandra	VeriSign

Page 65: [4] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [5] Change Anthony Nadalin 9/10/2005 5:48:00 PM
 Formatted Table

Page 65: [6] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [7] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [8] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [9] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [10] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [11] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [12] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [13] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [14] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [15] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [16] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [17] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [18] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [19] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [20] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM
 Tabs: Not at 3"

Page 65: [21] Formatted Anthony Nadalin 10/11/2005 9:47:00 AM

Tabs: Not at 3"

Page 65: [22] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [23] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [24] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [25] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [26] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [27] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [28] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [29] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [30] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [31] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [32] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [33] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [34] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [35] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [36] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [37] Formatted	Anthony Nadalin	10/11/2005 9:47:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [38] Formatted	Anthony Nadalin	10/11/2005 9:48:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [39] Formatted	Anthony Nadalin	10/11/2005 9:48:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [40] Formatted	Anthony Nadalin	10/11/2005 9:48:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [41] Formatted	Anthony Nadalin	10/11/2005 9:48:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [42] Formatted	Anthony Nadalin	10/11/2005 9:48:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [43] Formatted	Anthony Nadalin	10/11/2005 9:48:00 AM
-------------------------	-----------------	-----------------------

Tabs: Not at 3"

Page 65: [44] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 65: [45] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 1: [46] Deleted 28	Anthony Nadalin	9/3/2005 12:43:00 PM
Page 1: [46] Deleted June	Anthony Nadalin	9/3/2005 12:43:00 PM
Page 66: [47] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [48] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [49] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [50] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [51] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [52] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [53] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [54] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [55] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [56] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [57] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [58] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [59] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [60] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:44:00 AM
Page 66: [61] Change Formatted Table	Anthony Nadalin	9/10/2005 5:48:00 PM
Page 66: [62] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:44:00 AM
Page 66: [63] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:44:00 AM
Page 66: [64] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:44:00 AM
Page 66: [65] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:44:00 AM

Page 66: [66] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [67] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [68] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [69] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [70] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [71] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [72] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [73] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [74] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [75] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [76] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [77] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [78] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [79] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:48:00 AM
Page 66: [80] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [81] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [82] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [83] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [84] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [85] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [86] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [87] Formatted Tabs: Not at 3"	Anthony Nadalin	10/11/2005 9:49:00 AM
Page 66: [88] Formatted	Anthony Nadalin	10/11/2005 9:49:00 AM

Tabs: Not at 3"

Page 66: [89] Formatted **Anthony Nadalin** **10/11/2005 9:49:00 AM**

Tabs: Not at 3"

Page 66: [90] Formatted **Anthony Nadalin** **10/11/2005 9:49:00 AM**

Tabs: Not at 3"

Page 66: [91] Formatted **Anthony Nadalin** **10/11/2005 9:49:00 AM**

Tabs: Not at 3"

Page 1: [92] Deleted **Anthony Nadalin** **9/3/2005 12:43:00 PM**

28

Page 1: [92] Deleted **Anthony Nadalin** **9/3/2005 12:43:00 PM**

June

Page 68: [93] Deleted **Anthony Nadalin** **8/8/2005 11:21:00 AM**

WGD 1.1	2005-02-14	Anthony Nadalin	Issues 250, 351, 352
WGD 1.1	2005-03-22	Anthony Nadalin	Issues 310, 373, 374
WGD 1.1	2005-05-11	Anthony Nadalin	Issues 390, 84
WGD 1.1	2005-05-17	Anthony Nadalin	Formatting Issues
WGD 1.1	2005-06-14	Anthony Nadalin	Issues 400, mustUnderstand