

2002
PC Magazine Award
for Technical Excellence



FINALIST
OASIS WS-Security
OASIS

1 OASIS

2 **Web Services Security:**
3 **SOAP Message Security 1.0**

4 **Monday, 29 December 2003**

Deleted: 15

Deleted: December

Deleted: 3

5 **Document identifier:**

6 {draft}-{WSS: SOAP Message Security }-{1.0} (Word) (PDF)

7 **Location:**

8 <http://www.docs.oasis-open.org/wss/2003/12/oasis-####-wss-soap-message-security-1.0>

9 <http://www.oasis-open.org/committees/documents.php>

10 **Editors:**

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

11 **Contributors:**

Gene	Thurston	AmberPoint
Frank	Siebenlist	Argonne National Lab
Merlin	Hughes	Baltimore Technologies
Irving	Reid	Baltimore Technologies
Peter	Dapkus	BEA
Hal	Lockhart	BEA
Symon	Chang	CommerceOne
Thomas	DeMartini	ContentGuard
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard

Formatted Table

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

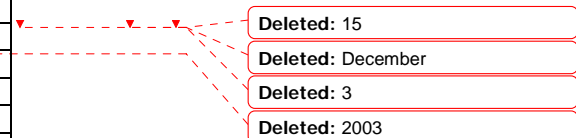
WSS: SOAP Message Security

Copyright © OASIS Open 2002-2003. All Rights Reserved.

29 December 2003

Page 1 of 60

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
Bob	Blakley	IBM
Joel	Farrell	IBM
Satoshi	Hada	IBM
Maryann	Hondo	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	Koneremann	Microsoft
Chris	Kurt	Microsoft
Brian	LaMacchia	Microsoft
Paul	Leach	Microsoft
John	Manferdell	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
Steve	Anderson	OpenNetwork (Sec)



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
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
John	Weiland	US Navy
Phillip	Hallam-Baker	VeriSign
Mark	Hays	Verisign
Hemma	Prafullchandra	VeriSign

12

13 **Abstract:**

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

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

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

26 **Status:**

27 This is a technical committee document submitted for consideration by the OASIS Web
 28 Services Security (WSS) technical committee. Please send comments to the editors. If
 29 you are on the wss@lists.oasis-open.org list for committee members, send comments
 30 there. If you are not on that list, subscribe to the wss-comment@lists.oasis-open.org list
 31 and send comments there. To subscribe, send an email message to wss-comment-
 32 request@lists.oasis-open.org with the word "subscribe" as the body of the message. For
 33 patent disclosure information that may be essential to the implementation of this
 34 specification, and any offers of licensing terms, refer to the Intellectual Property Rights
 35 section of the OASIS Web Services Security Technical Committee (WSS TC) web page

- Deleted: 15
- Deleted: December
- Deleted: 3
- Deleted: 2003

WSS: SOAP Message Security

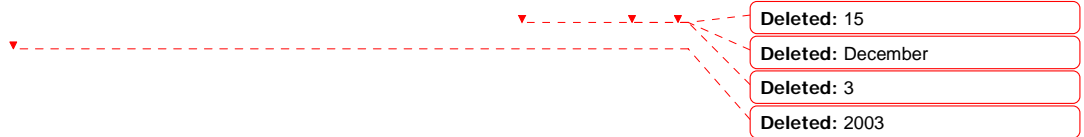
Copyright © OASIS Open 2002-2003. All Rights Reserved.

29 December 2003

Page 3 of 60

36
37

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>.



103 1 Introduction

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

Deleted: SAO

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

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

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

Formatted: Font color: Auto

122 1.1 Goals and Requirements

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

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

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

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

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

Deleted: 15

136 1.1.1 Requirements

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

Deleted: December

Deleted: 3

Deleted: 2003

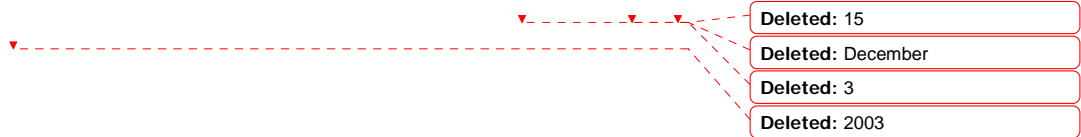
- 139 • Multiple security token formats
- 140 • Multiple trust domains
- 141 • Multiple signature formats

- 142 • Multiple encryption technologies
- 143 • End-to-end message content security and not just transport-level security

144 1.1.2 Non-Goals

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

- 146 • Establishing a security context or authentication mechanisms.
- 147 • Key derivation.
- 148 • Advertisement and exchange of security policy.
- 149 • How trust is established or determined.
- 150 • Non-repudiation.
- 151



2 Notations and Terminology

152

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

2.1 Notational Conventions

154

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

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

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

167 This specification is designed to work with the general SOAP message structure and message
168 processing model, and should be applicable to any version of SOAP, if future versions of SOAP
169 are backwards compatible. The current SOAP 1.2 namespace URI is used herein to provide
170 detailed examples, but there is no intention to limit the applicability of this specification to a single
171 version of SOAP.

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

2.2 Namespaces

173

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

176

```
177     http://www.docs.oasis-open.org/wss/2003/12/oasis-####-wss-  
178     wssecurity-secext-1.0.xsd  
179     http://www.docs.oasis-open.org/wss/2003/12/oasis-####-wss-  
180     wssecurity-utility-1.0.xsd
```

181

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

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

189

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#
wsse	http://www.docs.oasis-open.org/wss/2003/12/oasis-####-wss-wssecurity-secext-1.0.xsd
wsu	http://www.docs.oasis-open.org/wss/2003/12/oasis-####-wss-wssecurity-utility-1.0.xsd

Formatted Table

Field Code Changed

Field Code Changed

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

191 2.3 Acronyms and Abbreviations

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

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

Formatted Table

193 2.4 Terminology

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

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

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

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

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

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

Deleted: the basic

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

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

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

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

Deleted: the

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

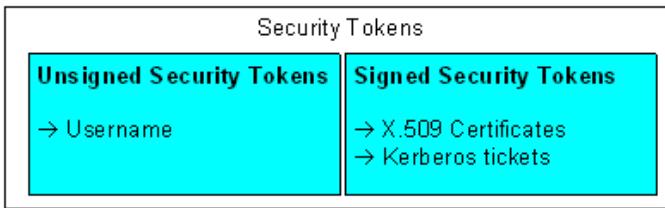
Deleted: since it was signed by

Deleted: .

Deleted: ¶

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

224 **Security Token** – A *security token* represents a collection (one or more) of claims.
 225



226
227

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

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

232
233
234

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

235

3 Message Protection Mechanisms

236
237

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

238
239
240

- the message could be modified or read by antagonists or
- an antagonist could send messages to a service that, while well-formed, lack appropriate security claims to warrant processing.

241

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

242

3.1 Message Security Model

243
244

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

245
246
247

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.

248
249
250
251
252

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.

253
254

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.

255
256
257
258

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.

262

3.2 Message Protection

263
264
265
266

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).

267
268
269
270

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

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

271 Message confidentiality leverages XML Encryption [XMLENC] in conjunction with security tokens
272 to keep portions of a SOAP message confidential. The encryption mechanisms are designed to
273 support additional encryption processes and operations by multiple SOAP roles.
274 This document defines syntax and semantics of signatures with `<wss:Security>` element.
275 This document does not specify any signature appearing outside of `<wss:Security>`
276 element.

Deleted: n

277 3.3 Invalid or Missing Claims

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

285 3.4 Example

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

```
292  
293 (001) <?xml version="1.0" encoding="utf-8"?>  
294 (002) <S:Envelope xmlns:S11="..."  
295         xmlns:ds="http://www.w3.org/2000/09/xmldsig#">  
296 (003)   <S:Header>  
297 (004)     <wss:Security  
298         xmlns:wss="...">  
299 (005)       <xxx:CustomToken wsu:Id="MyID"  
300         xmlns:xxx="http://fabrikam123/token">  
301           FHUIORv...  
302         </xxx:CustomToken>  
303         <ds:Signature>  
304           <ds:SignedInfo>  
305             <ds:CanonicalizationMethod  
306               Algorithm=  
307                 "http://www.w3.org/2001/10/xml-exc-c14n#" />  
308 (011)           <ds:SignatureMethod  
309               Algorithm=  
310                 "http://www.w3.org/2000/09/xmldsig#hmac-sha1" />  
311 (012)           <ds:Reference URI="#MsgBody">  
312 (013)             <ds:DigestMethod  
313                 Algorithm=  
314                 "http://www.w3.org/2000/09/xmldsig#sha1" />  
315 (014)             <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
```

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

Formatted: German Germany

```

316 | (015)         </ds:Reference>
317 | (016)         </ds:SignedInfo>
318 | (017)         <ds:SignatureValue>DjBchm5gK...</ds:SignatureValue>
319 | (018)         <ds:KeyInfo>
320 | (019)           <wsse:SecurityTokenReference>
321 | (020)           <wsse:Reference URI="#MyID"/>
322 | (021)         </wsse:SecurityTokenReference>
323 | (022)         </ds:KeyInfo>
324 | (023)         </ds:Signature>
325 | (024)         </wsse:Security>
326 | (025)         </S:Header>
327 | (026)         <S:Body wsu:Id="MsgBody">
328 | (027)           <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
329 | (028)             QQQ
330 | (029)           </tru:StockSymbol>
331 | (028)         </S:Body>
332 | (029) </S:Envelope>
333 |

```

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

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

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

340 Lines (008) to (035) specify a digital signature. This signature ensures the integrity of the signed
341 elements. The signature uses the XML Signature specification identified by the ds namespace
342 declaration in Line (002).

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

344 Line (010) specifies how to canonicalize (normalize) the data that is being signed. Lines (012) to
345 (015) select the elements that are signed and how to digest them. Specifically, line (012)

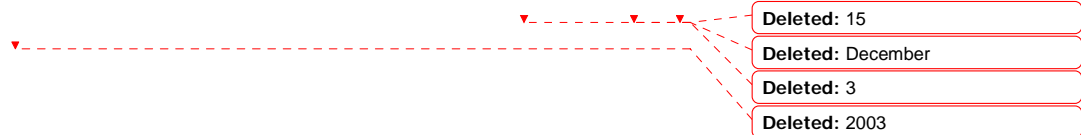
346 indicates that the <S:Body> element is signed. In this example only the message body is
347 signed; typically all critical elements of the message are included in the signature (see the
348 Extended Example below).

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

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

354 Lines (026) to (028) contain the *body* (payload) of the SOAP message.

355 |



356

4 ID References

357
358
359
360
361
362
363
364
365
366
367
368
369
370
371

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

- Local ID attributes on XML Signature elements
- Local ID attributes on XML Encryption elements
- Global *wsu:Id* attributes (described below) on elements
- In addition, when signing a part of an envelope such as the body, it is RECOMMENDED that an ID reference is used instead of a more general transformation, especially XPath [XPath]. This is to simplify processing.

372

4.1 Id Attribute

373
374
375
376
377
378
379
380
381
382
383
384
385
386

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

387

4.2 Id Schema

388
389
390
391
392
393

To simplify the processing for intermediaries and recipients, a common attribute is defined for identifying an element. This attribute utilizes the XML Schema ID type and specifies a common attribute for indicating this information for elements. The syntax for this attribute is as follows:

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

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419

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.

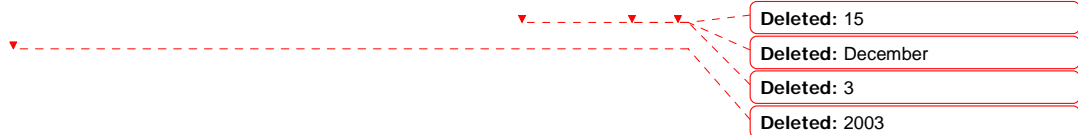
This specification does not specify how this attribute will be used and it is expected that other specifications MAY add additional semantics (or restrictions) for their usage of this attribute.

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 {name} 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.



420

5 Security Header

421 The `<wsse:Security>` header block provides a mechanism for attaching security-related
 422 information targeted at a specific recipient 'in the form of a' SOAP role. This may be either the
 423 ultimate recipient of the message or an intermediary. Consequently, elements of this type may
 424 be present multiple times in a SOAP message. An active intermediary on the message path MAY
 425 add one or more new sub-elements to an existing `<wsse:Security>` header block if they are
 426 targeted for its SOAP node or it MAY add one or more new headers for additional targets.
 427 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted
 428 for separate recipients. However, only one `<wsse:Security>` header block MAY omit the
 429 `S:role` attribute and no two `<wsse:Security>` header blocks MAY have the same value for
 430 `S:role`. Message security information targeted for different recipients MUST appear in different
 431 `<wsse:Security>` header blocks. The `<wsse:Security>` header block without a specified
 432 `S:role` MAY be processed by anyone, but MUST NOT be removed prior to the final destination or
 433 endpoint.
 434 As elements are added to a `<wsse:Security>` header block, they SHOULD be prepended to
 435 the existing elements. As such, the `<wsse:Security>` header block represents the signing and
 436 encryption steps the message producer took to create the message. This prepending rule
 437 ensures that the receiving application can process sub-elements in the order they appear in the
 438 `<wsse:Security>` header block, because there will be no forward dependency among the sub-
 439 elements. Note that this specification does not impose any specific order of processing the sub-
 440 elements. The receiving application can use whatever order is required.
 441 When a sub-element refers to a key carried in another sub-element (for example, a signature
 442 sub-element that refers to a binary security token sub-element that contains the X.509 certificate
 443 used for the signature), the key-bearing element SHOULD be ordered to precede the key-using
 444 Element:

Deleted: .

445
446
447
448
449
450
451
452
453
454
455

```

446 <S:Envelope>
447   <S:Header>
448     ...
449     <wsse:Security S:role="..." S:mustUnderstand="...">
450     ...
451   </wsse:Security>
452   ...
453 </S:Header>
454 ...
455 </S:Envelope>

```

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

456
457
458
459
460
461
462

457 The following describes the attributes and elements listed in the example above:
 458 `/wsse:Security`
 459 This is the header block for passing security-related message information to a recipient.
 460 `/wsse:Security/@S:role`
 461 This attribute allows a specific SOAP role to be identified. This attribute is optional;
 462 however, no two instances of the header block may omit a role or specify the same role.

463 /wsse:Security/{any}
 464 This is an extensibility mechanism to allow different (extensible) types of security
 465 information, based on a schema, to be passed.
 466 /wsse:Security/@{any}
 467 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 468 added to the header.
 469 All compliant implementations MUST be able to process a <wsse:Security> element.
 470 All compliant implementations MUST declare which profiles they support and MUST be able to
 471 process a <wsse:Security> element including any sub-elements which may be defined by that
 472 profile. It is RECCOMENDED that undefined elements within the <wsse:Security> header not
 473 be processed.
 474 The next few sections outline elements that are expected to be used within a <wsse:Security>
 475 header.
 476 When a <wsse: Security> header includes a mustUnderstand="true" attribute:
 477 • The receiver must generate a SOAP fault if does not implement the WSS: SOAP
 478 Message Security specification corresponding to the namespace. Implementation means
 479 ability to interpret the schema as well as follow the required processing rules specified in
 480 WSS: SOAP Message Security.
 481 • The receiver must generate a fault if unable to interpret or process security tokens
 482 contained in the <wsse:Security> header block according to the corresponding WSS:
 483 SOAP Message Security token profiles.
 484 • Receivers MAY ignore elements or extensions within the <wsse:Security> element,
 485 based on local security policy.

Formatted: Bullets and Numbering

Deleted: 15
 Deleted: December
 Deleted: 3
 Deleted: 2003

486

6 Security Tokens

487
488

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

Deleted: be

489

6.1 Attaching Security Tokens

490
491
492
493
494

This specification defines the `<wsse:Security>` header as a mechanism for conveying security information with and about a SOAP message. This header is, by design, extensible to support many types of security information. For security tokens based on XML, the extensibility of the `<wsse:Security>` header allows for these security tokens to be directly inserted into the header.

495

6.1.1 Processing Rules

496
497
498
499

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

500

6.1.2 Subject Confirmation

501
502
503

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

504

6.2 User Name Token

505

6.2.1 Usernames

506
507
508

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

509
510
511
512
513

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

Formatted: English U.S.

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

514
515
516
517
518

The following describes the attributes and elements listed in the example above:
`/wsse:UsernameToken`
 This element is used to represent a claimed identity.
`/wsse:UsernameToken/@wsu:Id`
 A string label for this security token.

519 `/wsse:UsernameToken/Username`
 520 This required element specifies the claimed identity.
 521 `/wsse:UsernameToken/Username/@{any}`
 522 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 523 added to the `<wsse:Username>` element.
 524 `/wsse:UsernameToken/{any}`
 525 This is an extensibility mechanism to allow different (extensible) types of security
 526 information, based on a schema, to be passed.
 527 `/wsse:UsernameToken/@{any}`
 528 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 529 added to the `<wsse:UsernameToken>` element.
 530 All compliant implementations MUST be able to process a `<wsse:UsernameToken>`
 531 element.
 532 The following illustrates the use of this:

```

533 <S:Envelope xmlns:S11="..."
534   <S:Header>
535     ...
536     <wsse:Security>
537       <wsse:UsernameToken>
538         <wsse:Username>Zoe</wsse:Username>
539       </wsse:UsernameToken>
540     </wsse:Security>
541   </S:Header>
542   ...
543 </S:Envelope>
544
545
546
  
```

Formatted: English U.S.

547 6.3 Binary Security Tokens

548 6.3.1 Attaching Security Tokens

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

552 6.3.2 Encoding Binary Security Tokens

553 Binary security tokens (e.g., X.509 certificates and Kerberos [KERBEROS] tickets) or other non-
 554 XML formats require a special encoding format for inclusion. This section describes a basic
 555 framework for using binary security tokens. Subsequent specifications MUST describe the rules
 556 for creating and processing specific binary security token formats.
 557 The `<wsse:BinarySecurityToken>` element defines two attributes that are used to interpret
 558 it. The `ValueType` attribute indicates what the security token is, for example, a Kerberos ticket.
 559 The `EncodingType` tells how the security token is encoded, for example `Base64Binary`.
 560 The following is an overview of the syntax:

Deleted: 15
 Deleted: December
 Deleted: 3
 Deleted: 2003

561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584

```
<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**.

Deleted: qualified nam

Deleted: This attribute value is extensible using XML namespaces [XML-ns].

/wsse:BinarySecurityToken/@EncodingType

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

Deleted: QName

Deleted: wsse:Base64Binary

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

Deleted: QName

Deleted: wsse:

585
586
587
588
589
590
591
592
593
594
595
596
597

/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.

When a `<wsse:BinarySecurityToken>` is included in a signature—that is, it is referenced from a `<ds:Signature>` element—care should be taken so that the canonicalization algorithm (e.g., Exclusive XML Canonicalization [EXC-C14N]) does not allow unauthorized replacement of namespace prefixes of the QNames used in the attribute or element values. In particular, it is **RECOMMENDED** that these namespace prefixes be declared within the `<wsse:BinarySecurityToken>` element if this token does not carry the validating key (and consequently it is not cryptographically bound to the signature).

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

Deleted: For example, if we wanted to sign the previous example, we need to include the consumed namespace definitions. ¶ In the following example, a custom `ValueType` is used. Consequently, the namespace definition for this `ValueType` is included in the `<wsse:BinarySecurityToken>` element. Note that the definition of `wsse` is also included as it is used for the encoding type and the element.

598 **6.4 XML Tokens**

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

601 **6.4.1 Identifying and Referencing Security Tokens**

602 This specification also defines multiple mechanisms for identifying and referencing security
603 tokens using the *wsu:id* attribute and the `<wsse:SecurityTokenReference>` element (as well
604 as some additional mechanisms). Please refer to the specific profile documents for the
605 appropriate reference mechanism. However, specific extensions MAY be made to the
606 `wsse:SecurityTokenReference` element.
607

```
Deleted: ¶  
<wsse:BinarySecurityToken ¶  
  xmlns:wsse="..." ¶  
  wsu:Id="myToken" ¶  
  ValueType="x:MyType" ¶  
  xmlns:x="http://www.fabrikam ¶  
  123.com/x"¶  
  EncodingType="wsse:Base64Bin ¶  
  ary">¶  
  MIEZzCCA9CgAwIBAgIQEmtJZc0. ¶  
  ..¶  
</wsse:BinarySecurityToken>¶
```

Formatted: Bullets and Numbering

- Deleted: 15
- Deleted: December
- Deleted: 3
- Deleted: 2003

608

7 Token References

609

This chapter discusses and defines mechanisms for referencing security tokens.

610

7.1 SecurityTokenReference Element

611

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

613

614

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

618

619

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

620

621

The following illustrates the syntax of this element:

622

623

624

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

625

626

627

The following describes the elements defined above:

628

`/wsse:SecurityTokenReference`

629

This element provides a reference to a security token.

630

`/wsse:SecurityTokenReference/@wsu:Id`

631

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 `<Reference>` element within the `<SecurityTokenReference>` element.

632

633

634

635

`/wsse:SecurityTokenReference/@wsse:Usage`

636

This optional attribute is used to type the usage of the `<SecurityToken>`. Usages are specified using [URIs](#) and multiple usages **MAY** be specified using XML list semantics.

637

638

[No usages are defined by this specification.](#)

639

`/wsse:SecurityTokenReference/{any}`

640

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

641

642

`/wsse:SecurityTokenReference/@{any}`

643

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

644

645

All compliant implementations **MUST** be able to process a

646

`<wsse:SecurityTokenReference>` element.

647

- Deleted: QName
- Deleted: ¶
- Deleted: QName (... [3])
- Deleted: 15
- Deleted: December
- Deleted: 3
- Deleted: 2003

648 This element can also be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to
 649 retrieve the key information from a security token placed somewhere else. In particular, it is
 650 RECOMMENDED, when using XML Signature and XML Encryption, that a
 651 `<wsse:SecurityTokenReference>` element be placed inside a `<ds:KeyInfo>` to reference
 652 the security token used for the signature or encryption.
 653 There are several challenges that implementations face when trying to interoperate. Processing
 654 the IDs and references requires the recipient to *understand* the schema. This may be an
 655 expensive task and in the general case impossible as there is no way to know the "schema
 656 location" for a specific namespace URI. As well, the primary goal of a reference is to uniquely
 657 identify the desired token. ID references are, by definition, unique by XML. However, other
 658 mechanisms such as "principal name" are not required to be unique and therefore such
 659 references may be not unique.
 660 The following list provides a list of the specific reference mechanisms defined in WSS: SOAP
 661 Message Security in preferred order (i.e., most specific to least specific):

- 662 • **Direct References** – This allows references to included tokens using URI fragments and
 663 external tokens using full URIs.
- 664 • **Key Identifiers** – This allows tokens to be referenced using an opaque value that
 665 represents the token (defined by token type/profile).
- 666 • **Key Names** – This allows tokens to be referenced using a string that matches an identity
 667 assertion within the security token. This is a subset match and may result in multiple
 668 security tokens that match the specified name.
- 669 • **Embedded References** - This allows tokens to be embedded (as opposed to a pointer
 670 to a token that resides elsewhere).

671 7.2 Direct References

672 The `<wsse:Reference>` element provides an extensible mechanism for directly referencing
 673 security tokens using URIs.
 674 The following illustrates the syntax of this element:

```
675 <wsse:SecurityTokenReference wsu:Id="...">
676   <wsse:Reference URI="..." ValueType="..." />
677 </wsse:SecurityTokenReference>
```

679 The following describes the elements defined above:

681 `/wsse:SecurityTokenReference/Reference`

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

683 `/wsse:SecurityTokenReference/Reference/@URI`

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

686 `/wsse:SecurityTokenReference/Reference/@ValueType`

687 This optional attribute specifies a URI that is used to identify the type of token being
 688 referenced. This specification does not define any processing rules around the usage of
 689 this attribute, however, specifications for individual token types MAY define specific
 690 processing rules and semantics around the value of the URI and how it SHALL be

- Deleted: 15
- Deleted: December
- Deleted: 3
- Deleted: 2003
- Deleted: QName
- Deleted: (see
<wsse:BinarySecurityToken>)

691 interpreted. If this attribute is not present, the URI MUST be processed as a normal URI.
 692 The usage of `ValueType` is RECOMMENDED for [references with](#) local URIs.
 693 `/wsse:SecurityTokenReference/Reference/{any}`
 694 This is an extensibility mechanism to allow different (extensible) types of security
 695 references, based on a schema, to be passed.
 696 `/wsse:SecurityTokenReference/Reference/@{any}`
 697 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
 698 added to the header.

699 The following illustrates the use of this element:

700
701
702
703
704
705

```
<wsse:SecurityTokenReference
  xmlns:wsse="...">
  <wsse:Reference
    URI="http://www.fabrikam123.com/tokens/Zoe"/>
</wsse:SecurityTokenReference>
```

706 7.3 Key Identifiers

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

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

716 The processing model assumes that the key identifier for a security token is constant.
 717 Consequently, processing a key identifier is simply looking for a security token whose key
 718 identifier matches a given specified constant.

719 The following is an overview of the syntax:

720
721
722
723
724
725
726
727

```
<wsse:SecurityTokenReference>
  <wsse:KeyIdentifier wsu:Id="..."
    ValueType="..."
    EncodingType="...">
    ...
  </wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
```

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

730 `/wsse:SecurityTokenReference/KeyIdentifier`

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

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

733 An optional string label for this identifier.

734 `/wsse:SecurityTokenReference/KeyIdentifier/@ValueType`

Formatted: ElementDesc Char Char,
Font: (Default) Courier New,
Complex Script Font: Courier New

Formatted: ElementDesc Char Char,
Font: (Default) Courier New,
Complex Script Font: Courier New

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

735 The optional `ValueType` attribute is used to indicate the type of `KeyIdentifier` being
 736 used. Each specific token profile specifies the `KeyIdentifier` types that may be used
 737 to refer to tokens of that type. It also specifies the critical semantics of the identifier, such
 738 as whether the `KeyIdentifier` is unique to the key or the token. If no value is specified
 739 then the key identifier will **be interpreted** in an application-specific manner.

Formatted: Font: (Default) Courier
 New, Complex Script Font: Courier
 New

Deleted: be interpreted

740 `/wsse:SecurityTokenReference/KeyIdentifier/@EncodingType`

741 The optional `EncodingType` attribute is used to indicate, using a [URI](#), the encoding
 742 format of the `KeyIdentifier` (`#Base64Binary`). The base values defined in this
 743 specification are used ([Note that URI fragments are relative to this document's URI](#)):
 744

Deleted: QName

Deleted: e.g., wsse:

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

Deleted: QName

Deleted: wsse:

745
 746 `/wsse:SecurityTokenReference/KeyIdentifier/@{any}`

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

749 7.4 Embedded References

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

753 The following is an overview of the syntax:

```
754
755 <wsse:SecurityTokenReference>
756   <wsse:Embedded wsu:Id="...">
757     ...
758   </wsse:Embedded>
759 </wsse:SecurityTokenReference>
```

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

761 `/wsse:SecurityTokenReference/Embedded`

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

764 `/wsse:SecurityTokenReference/Embedded/@wsu:Id`

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

767 `/wsse:SecurityTokenReference/Embedded/{any}`

768 This is an extensibility mechanism to allow any security token, based on schemas, to be
 769 embedded.

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

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

773 The following example illustrates embedding a SAML assertion:
 774

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791

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

792 7.5 ds:KeyInfo

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

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

799
800
801
802

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

803 7.6 Key Names

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

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

809
810

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

Formatted: ElementDesc Char Char,
Font: (Default) Courier New,
Complex Script Font: Courier New

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

811

8 Signatures

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

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

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

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

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

Deleted: and

838

8.1 Algorithms

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

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

843

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

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

844

845 [As well, the following table outlines additional algorithms that MAY be used:](#)

WSS: SOAP Message Security

Copyright © OASIS Open 2002-2003. All Rights Reserved.

29 December 2003

Page 28 of 60

Algorithm Type	Algorithm	Algorithm URI
Canonicalization	SOAP Message Normalization	http://www.w3.org/TR/2003/NOTE-soap12-n11n-20030328/

846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886

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, they SHOULD alter the order of the signature and encryption elements inside of the `<wsse:Security>` header. This order of elements represents order of operations. The XML Digital Signature WG has defined two canonicalization algorithms: XML Canonicalization and Exclusive XML Canonicalization. To prevent confusion, the first is also called Inclusive Canonicalization. Neither one solves all possible problems that can arise. The following informal discussion is intended to provide guidance on the choice of which one to use in particular circumstances. For a more detailed and technically precise discussion of these issues see: [XML-C14N] and [EXC-C14N].

There are two problems to be avoided. On the one hand, XML allows documents to be changed in various ways and still be considered equivalent. For example, duplicate namespace declarations can be removed or created. As a result, XML tools make these kinds of changes freely when processing XML. Therefore, it is vital that these equivalent forms match the same signature. On the other hand, if the signature simply covers something like `xx:foo`, its meaning may change if `xx` is redefined. In this case the signature does not prevent tampering. It might be thought that the problem could be solved by expanding all the values in line. Unfortunately, there are mechanisms like XPATH which consider `xx="http://example.com/"`; to be different from `yy="http://example.com/"`; even though both `xx` and `yy` are bound to the same namespace. The fundamental difference between the Inclusive and Exclusive Canonicalization is which namespace declarations which are placed in the output. Inclusive Canonicalization copies all the declarations that are currently in force, even if they are defined outside of the scope of the signature. It also copies any `xml:` attributes that are in force, such as `xml:lang` or `xml:base`. This guarantees that all the declarations you might make use of will be unambiguously specified. The problem with this is that if the signed XML is moved into another XML document which has other declarations, the Inclusive Canonicalization will copy them and the signature will be invalid. This can even happen if you simply add an attribute in a different namespace to the surrounding context.

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

Exclusive Canonicalization allows you to create a list of the namespaces that must be declared, so that it will pick up the declarations for the ones that are not visibly used. The only problem is

Formatted: ElementDesc Char Char, Font: (Default) Courier New, Complex Script Font: Courier New

Deleted: I

Field Code Changed

Field Code Changed

Deleted: are

Deleted: ¶
decla

Deleted: rations

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

887 that the software doing the signing must know what they are. In a typical SOAP software
888 environment, the security code will typically be unaware of all the namespaces being used by
889 the application in the message body that it is signing.
890 Exclusive Canonicalization is useful when you have a signed XML document that you wish to
891 insert into other XML documents. A good example is a signed SAML assertion which might be
892 inserted as a XML Token in the security header of various SOAP messages. The Issuer who
893 signs the assertion will be aware of the namespaces being used and able to construct the list.
894 The use of Exclusive Canonicalization will insure the signature verifies correctly every time.
895 Inclusive Canonicalization is useful in the typical case of signing part or all of the SOAP body in
896 accordance with this specification. This will insure all the declarations fall under the signature,
897 even though the code is unaware of what namespaces are being used. At the same time, it is
898 less likely that the signed data (and signature element) will be inserted in some other XML
899 document. Even if this is desired, it still may not be feasible for other reasons, for example there
900 may be Id's with the same value defined in both XML documents.
901 In other situations it will be necessary to study the requirements of the application and the
902 detailed operation of the canonicalization methods to determine which is appropriate.
903 This section is non-normative.
904

Deleted: t, T

905 8.2 Signing Messages

906 The <wss:Security> header block MAY be used to carry a signature compliant with the XML
907 Signature specification within a SOAP Envelope for the purpose of signing one or more elements
908 in the SOAP Envelope. Multiple signature entries MAY be added into a single SOAP Envelope
909 within one <wss:Security> header block. Producers SHOULD sign all important elements of
910 the message, and careful thought must be given to creating a signing policy that requires signing
911 of parts of the message that might legitimately be altered in transit.
912 SOAP applications MUST satisfy the following conditions:
913 A compliant implementation MUST be capable of processing the required elements defined in the
914 XML Signature specification.
915 To add a signature to a <wss:Security> header block, a <ds:Signature> element
916 conforming to the XML Signature specification MUST be prepended to the existing content of the
917 <wss:Security> header block, in order to indicate to the receiver the correct order of
918 operations. All the <ds:Reference> elements contained in the signature SHOULD refer to a
919 resource within the enclosing SOAP envelope as described in the XML Signature specification.
920 However, since the SOAP message exchange model allows intermediate applications to modify
921 the Envelope (add or delete a header block; for example), XPath filtering does not always result
922 in the same objects after message delivery. Care should be taken in using XPath filtering so that
923 there is no subsequent validation failure due to such modifications.
924 The problem of modification by intermediaries (especially active ones) is applicable to more than
925 just XPath processing. Digital signatures, because of canonicalization and digests, present
926 particularly fragile examples of such relationships. If overall message processing is to remain
927 robust, intermediaries must exercise care that the transformation algorithms used do not affect
928 the validity of a digitally signed component.

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

Deleted: their

Deleted: s

929 Due to security concerns with namespaces, this specification strongly RECOMMENDS the use of
930 the "Exclusive XML Canonicalization" algorithm or another canonicalization algorithm that
931 provides equivalent or greater protection.
932 For processing efficiency it is RECOMMENDED to have the signature added and then the
933 security token pre-pended so that a processor can read and cache the token before it is used.

934 8.3 Signing Tokens

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

944 This transform is specified by the URI `http://schemas.xmlsoap.org/2003/06/STR-`
945 `Transform` and when applied to a <SecurityTokenReference> element it means that the
946 output is the token referenced by the <SecurityTokenReference> element not the element
947 itself.
948 As an overview the processing model is to echo the input to the transform except when a
949 <SecurityTokenReference> element is encountered. When one is found, the element is not
950 echoed, but instead, it is used to locate the token(s) matching the criteria and rules defined by the
951 <SecurityTokenReference> element and echo it (them) to the output. Consequently, the
952 output of the transformation is the resultant sequence representing the input with any
953 <SecurityTokenReference> elements replaced by the referenced security token(s) matched.
954 The following illustrates an example of this transformation which references a token contained
955 within the message envelope:

956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974

```
...  
<wsse:SecurityTokenReference wsu:Id="Str1">  
  ...  
</wsse:SecurityTokenReference>  
...  
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">  
  <SignedInfo>  
    ...  
    <Reference URI="#Str1">  
      <Transforms>  
        <ds:Transform  
          Algorithm="http://schemas.xmlsoap.org/2003/06/STR-  
Transform">  
          <wsse:TransformationParameters>  
            <ds:CanonicalizationMethod  
              Algorithm="http://www.w3.org/TR/2001/REC-xml-  
c14n-20010315" />  
          </wsse:TransformationParameters>
```

- Deleted: 15
- Deleted: December
- Deleted: 3
- Deleted: 2003
- Formatted: French France
- Formatted: French France

975
976
977
978
979
980
981
982
983
984

```
</ds:Transform>
  <DigestMethod Algorithm=
    "http://www.w3.org/2000/09/xmldsig#sha1" />
  <DigestValue>...</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue></SignatureValue>
</Signature>
...
```

985
986
987
988
989
990
991

The following is a detailed specification of the transformation.
The algorithm is identified by the URI: <http://schemas.xmlsoap.org/2003/06/STR-Transform>

Transform Input:

- The input is a node set. If the input is an octet stream, then it is automatically parsed; cf. dsig.
- Transform Output:
- The output is an octet steam.

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

992

Syntax:

- The transform takes a single mandatory parameter, a ds:CanonicalizationMethod, which is used to serialize the input node set. Note, however, that the output may not be strictly in canonical form, per the canonicalization algorithm; however, the output is canonical, in the sense that it is unambiguous. [However, because of syntax requirements in the XML Signature definition, this parameter MUST be wrapped in a wsse:CanonicalizationParameters element.](#)

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

998

Processing Rules:

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

Formatted: ElementDesc Char Char, Complex Script Font: Courier New

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

1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018

Deleted: ¶

Deleted: The ValueType QName MUST use the same namespace prefix as the BinarySecurityToken element if the QName has the same namespace URI. Otherwise, it MUST use the namespace prefix x, or else the prefix y if Ri uses x. If no appropriate ValueType QName is known, then the transform MUST signal a failure.

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

Formatted: Bulleted + Level: 2 + Aligned at: 0.75" + Tab after: 1" + Indent at: 1"

1019 8.4 Signature Validation

1020 The validation of a <ds:Signature> element inside an <wsse:Security> header block
1021 SHALL fail if:
1022 the syntax of the content of the element does not conform to this specification, or
1023 the validation of the signature contained in the element fails according to the core validation of the
1024 XML Signature specification, or
1025 the application applying its own validation policy rejects the message for some reason (e.g., the
1026 signature is created by an untrusted key – verifying the previous two steps only performs
1027 cryptographic validation of the signature).
1028 if the validation of the signature element fails, applications MAY report the failure to the producer
1029 using the fault codes defined in Section 12 Error Handling.

Formatted: No bullets or numbering

1030 8.5 Example

1031 The following sample message illustrates the use of integrity and security tokens. For this
1032 example, only the message body is signed.

```
1033 <?xml version="1.0" encoding="utf-8"?>
1034 <S:Envelope xmlns:S11="..."
1035   <S:Header>
1036     <wsse:Security>
1037       <wsse:BinarySecurityToken
1038         ValueType="...#X509v3"
1039         EncodingType="...#Base64Binary"
1040         wsu:Id="X509Token">
1041           MIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1042       </wsse:BinarySecurityToken>
1043       <ds:Signature>
1044         <ds:SignedInfo>
1045           <ds:CanonicalizationMethod Algorithm=
1046             "http://www.w3.org/2001/10/xml-exc-c14n#" />
1047           <ds:SignatureMethod Algorithm=
1048             "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
1049           <ds:Reference URI="#myBody">
1050             <ds:Transforms>
1051               <ds:Transform Algorithm=
1052                 "http://www.w3.org/2001/10/xml-exc-c14n#" />
1053             </ds:Transforms>
1054             <ds:DigestMethod Algorithm=
1055               "http://www.w3.org/2000/09/xmldsig#sha1" />
1056             <ds:DigestValue>EULddytSol...</ds:DigestValue>
1057           </ds:Reference>
1058         </ds:SignedInfo>
1059         <ds:SignatureValue>
1060           BL8jdfToEbl1/vXcMZNNjPOV...
1061         </ds:SignatureValue>
1062         <ds:KeyInfo>
1063           <wsse:SecurityTokenReference>
1064             <wsse:Reference URI="#X509Token" />
1065           </wsse:SecurityTokenReference>
1066         </ds:KeyInfo>
1067       </ds:Signature>
1068     </wsse:Security>
1069   </S:Header>
1070   <S:Body>
1071     ...
1072   </S:Body>
1073 </S:Envelope>
```

Deleted: wsse:

Deleted: wsse:

Deleted: 15

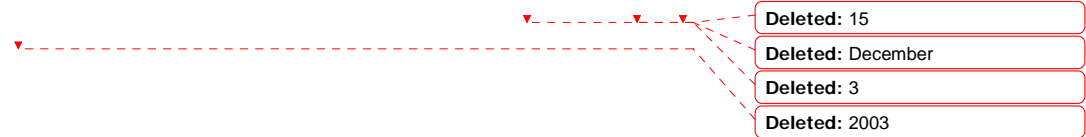
Deleted: December

Deleted: 3

Deleted: 2003

1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076

```
        </wsse:SecurityTokenReference>  
        </ds:KeyInfo>  
        </ds:Signature>  
    </wsse:Security>  
</S:Header>  
<S:Body wsu:Id="myBody">  
    <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">  
        QQQ  
    </tru:StockSymbol>  
</S:Body>  
</S:Envelope>
```



1077

9 Encryption

1078 This specification allows encryption of any combination of body blocks, header blocks, and any of
 1079 these sub-structures by either a common symmetric key shared by the producer and the recipient
 1080 or a symmetric key carried in the message in an encrypted form.
 1081 In order to allow this flexibility, this specification leverages the XML Encryption standard.
 1082 Specifically what this specification describes is how three elements (listed below and defined in
 1083 XML Encryption) can be used within the `<wsse:Security>` header block. When a producer or
 1084 an active intermediary encrypts portion(s) of a SOAP message using XML Encryption they MUST
 1085 prepend a sub-element to the `<wsse:Security>` header block. Furthermore, the encrypting
 1086 party MUST either prepend the sub-element to an existing `<wsse:Security>` header block for
 1087 the intended recipients or create a new `<wsse:Security>` header block and insert the sub-
 1088 element. The combined process of encrypting portion(s) of a message and adding one of these
 1089 sub-elements is called an encryption step hereafter. The sub-element MUST contain the
 1090 information necessary for the recipient to identify the portions of the message that it is able to
 1091 decrypt.
 1092 All compliant implementations MUST be able to support the XML Encryption standard.

Deleted:

1093

9.1 xenc:ReferenceList

1094 The `<xenc:ReferenceList>` element from XML Encryption MAY be used to create a manifest
 1095 of encrypted portion(s), which are expressed as `<xenc:EncryptedData>` elements within the
 1096 envelope. An element or element content to be encrypted by this encryption step MUST be
 1097 replaced by a corresponding `<xenc:EncryptedData>` according to XML Encryption. All the
 1098 `<xenc:EncryptedData>` elements created by this encryption step SHOULD be listed in
 1099 `<xenc:DataReference>` elements inside one or more `<xenc:ReferenceList>` element.
 1100 Although in XML Encryption, `<xenc:ReferenceList>` was originally designed to be used
 1101 within an `<xenc:EncryptedKey>` element (which implies that all the referenced
 1102 `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows
 1103 that `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>`
 1104 MAY be encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>`
 1105 within individual `<xenc:EncryptedData>`.
 1106 A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the
 1107 producer and the recipient use a shared secret key. The following illustrates the use of this sub-
 1108 element:

```

1109
1110 <S:Envelope
1111   <S:Envelope xmlns:S11="..."
1112     <S:Header>
1113       <wsse:Security>
1114         <xenc:ReferenceList>
1115           <xenc:DataReference URI="#bodyID" />
1116         </xenc:ReferenceList>

```

Deleted: 15
 Deleted: December
 Deleted: 3
 Deleted: 2003

```

1117     </wsse:Security>
1118 </S:Header>
1119 <S:Body>
1120     <xenc:EncryptedData Id="bodyID">
1121       <ds:KeyInfo>
1122         <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
1123       </ds:KeyInfo>
1124       <xenc:CipherData>
1125         <xenc:CipherValue>...</xenc:CipherValue>
1126       </xenc:CipherData>
1127     </xenc:EncryptedData>
1128   </S:Body>
1129 </S:Envelope>

```

1130 9.2 xenc:EncryptedKey

1131 When the encryption step involves encrypting elements or element contents within a SOAP
1132 envelope with a symmetric key, which is in turn to be encrypted by the recipient's key and
1133 embedded in the message, <xenc:EncryptedKey> MAY be used for carrying such an
1134 encrypted key. This sub-element SHOULD have a manifest, that is, an
1135 <xenc:ReferenceList> element, in order for the recipient to know the portions to be
1136 decrypted with this key. An element or element content to be encrypted by this encryption step
1137 MUST be replaced by a corresponding <xenc:EncryptedData> according to XML Encryption.
1138 All the <xenc:EncryptedData> elements created by this encryption step SHOULD be listed in
1139 the <xenc:ReferenceList> element inside this sub-element.
1140 This construct is useful when encryption is done by a randomly generated symmetric key that is
1141 in turn encrypted by the recipient's public key. The following illustrates the use of this element:

```

1142 <S:Envelope
1143   <S:Envelope xmlns:S11="..."
1144     <S:Header>
1145       <wsse:Security>
1146         <xenc:EncryptedKey>
1147           ...
1148           <ds:KeyInfo>
1149             <wsse:SecurityTokenReference>
1150               <ds:X509IssuerSerial>
1151                 <ds:X509IssuerName>
1152                   DC=ACMECorp, DC=com
1153                 </ds:X509IssuerName>
1154               <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
1155             </ds:X509IssuerSerial>
1156           </wsse:SecurityTokenReference>
1157         </ds:KeyInfo>
1158       </wsse:Security>
1159     </S:Header>
1160     <S:Body>
1161       <xenc:EncryptedData Id="bodyID">
1162         <xenc:CipherData>

```

1167
1168
1169
1170
1171
1172

```
<xenc:CipherValue>...</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</S:Body>
</S:Envelope>
```

1173 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in
1174 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that
1175 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

1176 9.3 Processing Rules

1177 Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the
1178 XML Encryption specification. An encrypted SOAP envelope MUST still be a valid SOAP
1179 envelope. The message creator MUST NOT encrypt the `<S:Envelope>`, `<S:Header>`, or
1180 `<S:Body>` elements but MAY encrypt child elements of either the `<S:Header>` and/or
1181 `<S:Body>` elements. Multiple steps of encryption MAY be added into a single `<Security>`
1182 header block if they are targeted for the same recipient.
1183 When an element or element content inside a SOAP envelope (e.g. the contents of the
1184 `<S:Body>` element) is to be encrypted, it MUST be replaced by an `<xenc:EncryptedData>`,
1185 according to XML Encryption and it SHOULD be referenced from the `<xenc:ReferenceList>`
1186 element created by this encryption step.

1187 9.3.1 Encryption

1188 The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1189 this specification are listed below (note that use of `<xenc:ReferenceList>` is
1190 RECOMMENDED).

- 1191 • Create a new SOAP envelope.
- 1192 • Create a `<Security>` header
- 1193 • When an EncryptedKey is used, create a `<xenc:EncryptedKey>` sub-element of the
1194 `<wsse:Security>` element. This `<xenc:EncryptedKey>` sub-element SHOULD contain
1195 an `<xenc:ReferenceList>` sub-element, containing a `<xenc:DataReference>` to
1196 each `<xenc:EncryptedData>` element that was encrypted using that key.
- 1197 • Locate data items to be encrypted, i.e., XML elements, element contents within the target
1198 SOAP envelope.
- 1199 • Encrypt the data items as follows: For each XML element or element content within the
1200 target SOAP envelope, encrypt it according to the processing rules of the XML
1201 Encryption specification. Each selected original element or element content MUST be
1202 removed and replaced by the resulting `<xenc:EncryptedData>` element.
- 1203 • The optional `<ds:KeyInfo>` element in the `<xenc:EncryptedData>` element MAY
1204 reference another `<ds:KeyInfo>` element. Note that if the encryption is based on an
1205 attached security token, then a `<SecurityTokenReference>` element SHOULD be
1206 added to the `<ds:KeyInfo>` element to facilitate locating it.

Formatted: ElementDesc Char Char, Font: (Default) Courier New, Complex Script Font: Courier New

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

Formatted: ElementDesc Char Char, Font: (Default) Courier New, Complex Script Font: Courier New

Formatted: ElementDesc Char Char, Font: (Default) Courier New, Complex Script Font: Courier New

Formatted: ElementDesc Char Char, Font: (Default) Courier New, Complex Script Font: Courier New

Formatted: ElementDesc Char Char, Font: (Default) Courier New, Complex Script Font: Courier New

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

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

- 1207 | • Create an `<xenc:DataReference>` element referencing the generated
- 1208 | `<xenc:EncryptedData>` elements. Add the created `<xenc:DataReference>`
- 1209 | element to the `<xenc:ReferenceList>`.
- 1210 | • Copy all non-encrypted data.

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

1211 | 9.3.2 Decryption

1212 | On receiving a SOAP envelope containing encryption header elements, for each encryption
 1213 | header element the following general steps should be processed (non-normative):
 1214 | Identify any decryption keys that are in the recipient's possession, then identifying any message
 1215 | elements that it is able to decrypt.

1216 | Locate the `<xenc:EncryptedData>` items to be decrypted (possibly using the
 1217 | `<xenc:ReferenceList>`).

Formatted: No bullets or
 numbering

1218 | Decrypt them as follows:

- 1219 | • For each element in the target SOAP envelope, decrypt it according to the processing
- 1220 | rules of the XML Encryption specification and the processing rules listed above.
- 1221 | • If the decryption fails for some reason, applications MAY report the failure to the producer
- 1222 | using the fault code defined in Section 12 Error Handling of this specification.

Formatted: Bullets and Numbering

1223 | It is possible for overlapping portions of the SOAP message to be encrypted in such a way that
 1224 | they are intended to be decrypted by SOAP nodes acting in different Roles. In this case, the
 1225 | `ReferenceList` or `EncryptedKey` elements identifying these encryption operations will
 1226 | necessarily appear in different Security headers. Since SOAP does not provide any means of
 1227 | specifying the order in which different Roles will process their respective headers, this order is not
 1228 | specified by this specification and can only be determined by a prior agreement.

1229 | 9.4 Decryption Transformation

1230 | The ordering semantics of the `<wsse:Security>` header are sufficient to determine if
 1231 | signatures are over encrypted or unencrypted data. However, when a signature is included in
 1232 | one `<wsse:Security>` header and the encryption data is in another `<wsse:Security>`
 1233 | header, the proper processing order may not be apparent.

1234 | If the producer wishes to sign a message that MAY subsequently be encrypted by an
 1235 | intermediary then the producer MAY use the Decryption Transform for XML Signature to explicitly
 1236 | specify the order of decryption.
 1237 |

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1238

10 Security Timestamps

1239 It is often important for the recipient to be able to determine the *freshness* of security semantics.
 1240 In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.
 1241 This specification does not provide a mechanism for synchronizing time. The assumption is that
 1242 time is trusted or additional mechanisms, not described here, are employed to prevent replay.
 1243 This specification defines and illustrates time references in terms of the *dateTime* type defined in
 1244 XML Schema. It is RECOMMENDED that all time references use this type. It is further
 1245 RECOMMENDED that all references be in UTC time. Implementations MUST NOT generate time
 1246 instants that specify leap seconds. If, however, other time types are used, then the *ValueType*
 1247 attribute (described below) MUST be specified to indicate the data type of the time format.
 1248 Requestors and receivers SHOULD NOT rely on other applications supporting time resolution
 1249 finer than milliseconds.

1250 The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and
 1251 expiration times of the security semantics in a message.
 1252 All times SHOULD be in UTC format as specified by the XML Schema type (*dateTime*). It should
 1253 be noted that times support time precision as defined in the XML Schema specification.

1254 The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and
 1255 may only be present at most once per header (that is, per SOAP role).

1256 The ordering within the element is as illustrated below. The ordering of elements in the
 1257 `<wsu:Timestamp>` element is fixed and MUST be preserved by intermediaries.

1258 The schema outline for the `<wsu:Timestamp>` element is as follows:

1259

1260

1261

1262

1263

1264

```

<wsu:Timestamp wsu:Id="...">
  <wsu:Created ValueType="...">...</wsu:Created>
  <wsu:Expires ValueType="...">...</wsu:Expires>
  ...
</wsu:Timestamp>

```

1265

1266 The following describes the attributes and elements listed in the schema above:

1267

`/wsu:Timestamp`

1268

This is the element for indicating message timestamps.

1269

`/wsu:Timestamp/wsue:Created`

1270

This represents the creation time of the security semantics. This element is optional, but
 1271 can only be specified once in a `Timestamp` element. Within the SOAP processing
 1272 model, creation is the instant that the infoset is serialized for transmission. The creation
 1273 time of the message SHOULD NOT differ substantially from its transmission time. The
 1274 difference in time should be minimized.

1275

`/wsu:Timestamp/wsue:Created/@ValueType`

1276

This optional attribute specifies the type of the time data. This is specified as the XML

1277

Schema type. The default value is `xsd:dateTime`.

1278

`/wsu:Timestamp/wsue:Expires`

Formatted: ElementDesc Char Char,
 Font: (Default) Courier New,
 Complex Script Font: Courier New

Deleted: ¶
 To preserve overall integrity of each
`<wsu:Timestamp>` element, it is
 strongly RECOMMENDED that each
 SOAP role only create or update the
 appropriate `<wsu:Timestamp>`
 element destined to itself (that is, a
`<wsse:Security>` header whose
 actor/role is itself) and no other
`<wsu:Timestamp>` element.

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1279 This element represents the expiration of the security semantics. This is optional, but
 1280 can appear at most once in a `Timestamp` element. Upon expiration, the requestor
 1281 asserts that its security semantics are no longer valid. It is strongly RECOMMENDED
 1282 that recipients (anyone who processes this message) discard (ignore) any message
 1283 whose security semantics have passed their expiration. A Fault code
 1284 (`wsu:MessageExpired`) is provided if the recipient wants to inform the requestor that its
 1285 security semantics were expired. A service MAY issue a Fault indicating the security
 1286 semantics have expired.

1287 `/wsu:Timestamp/wsu:Expires/@ValueType`
 1288 This optional attribute specifies the type of the time data. This is specified as the XML
 1289 Schema type. The default value is `xsd:dateTime`.

1290 `/wsu:Timestamp/{any}`
 1291 This is an extensibility mechanism to allow additional elements to be added to the
 1292 element.

1293 `/wsu:Timestamp/@wsu:Id`
 1294 This optional attribute specifies an XML Schema ID that can be used to reference this
 1295 element (the timestamp). This is used, for example, to reference the timestamp in a XML
 1296 Signature.

1297 `/wsu:Timestamp/@{any}`
 1298 This is an extensibility mechanism to allow additional attributes to be added to the
 1299 element.

1300 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
 1301 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
 1302 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
 1303 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
 1304 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
 1305 judgment of the requestor's likely current clock time by means not described in this specification,
 1306 for example an out-of-band clock synchronization protocol. The recipient may also use the
 1307 creation time and the delays introduced by intermediate SOAP roles to estimate the degree of
 1308 clock skew.

1309 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```

1310
1311 <S:Envelope xmlns:S11="..."
1312 <S:Header>
1313 <wsse:Security>
1314 <wsu:Timestamp wsu:Id="timestamp">
1315 <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
1316 <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
1317 </wsu:Timestamp>
1318 ...
1319 </wsse:Security>
1320 ...
1321 </S:Header>
1322 <S:Body>
1323 ...
1324 </S:Body>
1325 </S:Envelope>
  
```



1326

11 Extended Example

1327 The following sample message illustrates the use of security tokens, signatures, and encryption.
 1328 For this example, the timestamp and the message body are signed prior to encryption. The
 1329 decryption transformation is not needed as the signing/encryption order is specified within the
 1330 <wsse:Security> header.

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

```

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

```

Comment: If this edit is accepted then the Created element should probably be all on one line. This throws off the line numbers. There doesn't seem to be an automatic way to renumber. If this has to be done by hand (renumber) I'll be happy to do the grunt work. I don't want to do it though if it's not necessary.

Deleted: wsse

Deleted: :

Deleted: wsse:

Deleted: wsse:

Deleted: wsse

Deleted: :

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

```

1373           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1374 (034)         </ds:Transforms>
1375 (035)         <ds:DigestMethod
1376           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1377 (036)         <ds:DigestValue>LyLsF094hPi4wPU...
1378 (037)         </ds:DigestValue>
1379 (038)         </ds:Reference>
1380 (039)         <ds:Reference URI="#body">
1381 (040)         <ds:Transforms>
1382 (041)         <ds:Transform
1383           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1384 (042)         </ds:Transforms>
1385 (043)         <ds:DigestMethod
1386           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1387 (044)         <ds:DigestValue>LyLsF094hPi4wPU...
1388 (045)         </ds:DigestValue>
1389 (046)         </ds:Reference>
1390 (047)         </ds:SignedInfo>
1391 (048)         <ds:SignatureValue>
1392 (049)           Hp1ZkmFZ/2kQLXDJbchm5gK...
1393 (050)         </ds:SignatureValue>
1394 (051)         <ds:KeyInfo>
1395 (052)           <wsse:SecurityTokenReference>
1396 (053)             <wsse:Reference URI="#X509Token" />
1397 (054)           </wsse:SecurityTokenReference>
1398 (055)           </ds:KeyInfo>
1399 (056)         </ds:Signature>
1400 (057)       </wsse:Security>
1401 (058)     </S:Header>
1402 (059)     <S:Body wsu:Id="body">
1403 (060)       <xenc:EncryptedData
1404           Type="http://www.w3.org/2001/04/xmenc#Element"
1405           wsu:Id="encl1">
1406 (061)       <xenc:EncryptionMethod
1407           Algorithm="http://www.w3.org/2001/04/xmenc#tripleDES-
1408 cbc" />
1409 (062)       <xenc:CipherData>
1410 (063)         <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1411 (064)         </xenc:CipherValue>
1412 (065)       </xenc:CipherData>
1413 (066)     </xenc:EncryptedData>
1414 (067)   </S:Body>
1415 (068) </S:Envelope>

```

1416
1417
1418
1419
1420
1421
1422

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

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

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

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

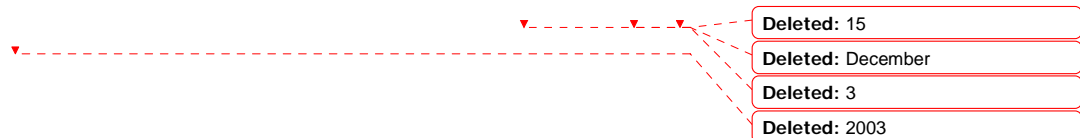
Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1423 Lines (010)-(012) specify a security token that is associated with the message. In this case, it
1424 specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64
1425 encoding of the certificate.
1426 Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a
1427 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
1428 encrypt the key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the
1429 symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines
1430 (023)-(025) identify the encryption block in the message that uses this symmetric key. In this
1431 case it is only used to encrypt the body (Id="enc1").
1432 Lines (027)-(056) specify the digital signature. In this example, the signature is based on the
1433 X.509 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039)
1434 references the message body.
1435 Lines (048)-(050) indicate the actual signature value – specified in Line (043).
1436 Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the X.509
1437 certificate included in the message. Line (053) provides a URI link to the Lines (010)-(012).
1438 The body of the message is represented by Lines (057)-(067).
1439 Lines (060)-(066) represent the encrypted metadata and form of the body using XML Encryption.
1440 Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line
1441 (061) specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the
1442 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
1443 key as the key references this encryption – Line (024).



1444

12 Error Handling

1445

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

1446

For example:

1447

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

1448

- Invalid or unauthenticated or unauthenticatable security token

1449

- Invalid signature

1450

- Decryption failure

1451

- Referenced security token is unavailable

1452

- Unsupported namespace

1453

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

1454

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

1455

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

1456

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

1457

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~~

1458

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

1459

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

1460

1461

1462

1463

1464

Deleted: :

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

1465

The "failure" class of errors are:

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

Deleted: 15

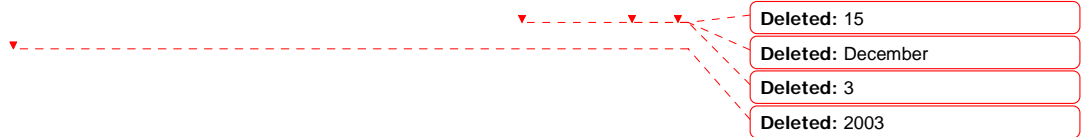
Deleted: December

Deleted: 3

Deleted: 2003

Referenced security token could not be retrieved

wsse:SecurityTokenUnavailable



1466
1467
1468
1469
1470
1471
1472
1473

1474

1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497

13 Security Considerations

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.

13.1 General Considerations

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.

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

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.

Formatted: Line spacing: single

Deleted: 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. ¶

¶
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. ¶
<#>freshness guarantee (e.g., the danger of replay, delayed messages and the danger of relying on timestamps assuming secure clock synchronization) ¶
<#>proper use of digital signature and encryption (signing/encrypting critical parts of the message, interactions between signatures and encryption), i.e., signatures on (content of) encrypted messages leak information when in plain-text) ¶
<#>protection of security tokens (integrity) ¶
<#>certificate verification (including revocation issues) ¶
<#>the danger of using passwords without outmost protection (i.e. dictionary attacks against passwords, replay, insecurity of password derived keys, ...)

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1498

13.2 Additional Considerations

Formatted: Bullets and Numbering

1499

13.2.1 Replay

1500

Digital signatures alone do not provide message authentication. One can record a signed message and resend it (a replay attack). It is strongly RECOMMENDED that messages include digitally signed elements to allow message recipients to detect replays of the message when the messages are exchanged via an open network. These can be part of the message or of the headers defined from other SOAP extensions. Four typical approaches are: Timestamp, Sequence Number, Expirations and Message Correlation. Signed timestamps MAY be used to keep track of messages (possibly by caching the most recent timestamp from a specific service) and detect replays of previous messages. It is RECOMMENDED that timestamps be cached for a given period of time, as a guideline, a value of five minutes can be used as a minimum to detect replays, and that timestamps older than that given period of time set be rejected in interactive scenarios.

1501

1502

1503

1504

1505

1506

1507

1508

1509

1510

Formatted: Bullets and Numbering

1511

13.2.2 Combining Security Mechanisms

1512

This specification defines the use of XML Signature and XML Encryption in SOAP headers. As one of the building blocks for securing SOAP messages, it is intended to be used in conjunction with other security techniques. Digital signatures need to be understood in the context of other security mechanisms and possible threats to an entity. Implementers should also be aware of all the security implications resulting from the use of digital signatures in general and XML Signature in particular. When building trust into an application based on a digital signature there are other technologies, such as certificate evaluation, that must be incorporated, but these are outside the scope of this document.

1513

1514

1515

1516

1517

1518

1519

1520

1521

1522

Formatted: Bullets and Numbering

1523

13.2.3 Challenges

1524

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

1525

1526

1527

Formatted: Bullets and Numbering

1528

13.2.4 Protecting Security Tokens and Keys

1529

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

1530

1531

1532

1533

1534

1535

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1536 prevent a different authority from issuing a token over the same key if the user can prove
1537 possession of the secret.
1538 The most straightforward counter to this attack is to insist that the token (or its unique identifying
1539 data) be included under the signature of the producer. If the nature of the application is such that
1540 the contents of the token are irrelevant, assuming it has been issued by a trusted authority, this
1541 attack may be ignored. However because application semantics may change over time, best
1542 practice is to prevent this attack.
1543 Requestors should use digital signatures to sign security tokens that do not include signatures (or
1544 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
1545 RECOMMENDED that all relevant and immutable message content be signed by the producer.
1546 Receivers SHOULD only consider those portions of the document that are covered by the
1547 producer's signature as being subject to the security tokens in the message. Security tokens
1548 appearing in <wsse:Security> header elements SHOULD be signed by their issuing authority
1549 so that message receivers can have confidence that the security tokens have not been forged or
1550 altered since their issuance. It is strongly RECOMMENDED that a message producer sign any
1551 <SecurityToken> elements that it is confirming and that are not signed by their issuing
1552 authority.
1553 When a requester provides, within the request, a Public Key to be used to encrypt the response,
1554 it is possible that an attacker in the middle may substitute a different Public Key, thus allowing the
1555 attacker to read the response. The best way to prevent this attack is to bind the encryption key in
1556 some way to the request. One simple way of doing this is to use the same key pair to sign the
1557 request as to encrypt the response. However, if policy requires the use of distinct key pairs for
1558 signing and encryption, then the Public Key provided in the request should be included under the
1559 signature of the request.

Formatted: Bullets and Numbering

1560 **13.2.5 Protecting Timestamps and Ids**

1561 In order to trust <wsu:Ids> and <wsu:Timestamp> elements, they SHOULD be signed using
1562 the mechanisms outlined in this specification. This allows readers of the IDs and timestamps
1563 information to be certain that the IDs and timestamps haven't been forged or altered in any way.
1564 It is strongly RECOMMENDED that IDs and timestamp elements be signed.

1565 This section is non-normative..
1566

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1567

14 Interoperability Notes

1568 Based on interoperability experiences with this and similar specifications, the following list
1569 highlights several common areas where interoperability issues have been discovered. Care
1570 should be taken when implementing to avoid these issues. It should be noted that some of these
1571 may seem "obvious", but have been problematic during testing.

- 1572 | • **Key Identifiers:** Make sure you understand the algorithm and how it is applied to security*
1573 | tokens.
- 1574 | • **EncryptedKey.** The `EncryptedKey` element from XML Encryption requires a Type
1575 | attribute whose value is one of a pre-defined list of values. Ensure that a correct value is
1576 | used.
- 1577 | • **Encryption Padding:** The XML Encryption random block cipher padding has caused
1578 | issues with certain decryption implementations; be careful to follow the specifications
1579 | exactly.
- 1580 | • **IDs:** The specification recognizes three specific ID elements: the global `wsu:Id` attribute ←
1581 | and the local `Id` attributes on XML Signature and XML Encryption elements (because the
1582 | latter two do not allow global attributes). If any other element does not allow global
1583 | attributes, it cannot be directly signed using an ID reference. Note that the global
1584 | attribute `wsu:Id` MUST carry the namespace specification.
- 1585 | • **Time Formats:** This specification uses a restricted version of the XML Schema
1586 | `dateTime` element. Take care to ensure compliance with the specified restrictions.
- 1587 | • **Byte Order Marker (BOM):** Some implementations have problems processing the BOM
1588 | marker. It is suggested that usage of this be optional.
- 1589 | • **SOAP, WSDL, HTTP:** Various interoperability issues have been seen with incorrect
1590 | SOAP, WSDL, and HTTP semantics being applied. Care should be taken to carefully
1591 | adhere to these specifications and any interoperability guidelines that are available.

1592 This section is non-normative.

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

Formatted: Bullets and Numbering

Formatted: ElementDesc Char Char,
Font: (Default) Courier New,
Complex Script Font: Courier New

Formatted: ElementDesc Char Char,
Font: (Default) Courier New,
Complex Script Font: Courier New

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1593

15 Privacy Considerations

1594 In the context of this specification, we are only concerned with potential privacy violation by the
1595 security elements defined here. Privacy of the content of the payload message is out of scope.
1596 Producers or sending applications should be aware that claims, as collected in security tokens,
1597 are typically personal information, and should thus only be sent according to the producer's
1598 privacy policies. Future standards may ~~allow privacy~~ obligations or restrictions to be added to this
1599 data. Unless such standards are used, the producer must ensure out-of-band that the recipient is
1600 bound to adhering to all restrictions associated with the data, and the recipient must similarly
1601 ensure by out-of-band means that it has the necessary consent for its intended processing of the
1602 data.
1603 If claim data are visible to intermediaries, then the policies must also allow the release to these
1604 intermediaries. As most personal information cannot be released to arbitrary parties, this will
1605 typically require that the actors are referenced in an identifiable way; such identifiable references
1606 are also typically needed to obtain appropriate encryption keys for the intermediaries.
1607 If intermediaries add claims, they should be guided by their privacy policies just like the original
1608 producers.
1609 Intermediaries may also gain traffic information from a SOAP message exchange, e.g., who
1610 communicates with whom at what time. Producers that use intermediaries should verify that
1611 releasing this traffic information to the chosen intermediaries conforms to their privacy policies.
1612 This section is non-normative.

Deleted: allow privacy

Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

16References

1613

- 1614 [GLOSS] Informational RFC 2828, "[Internet Security Glossary](#)," May 2000.
- 1615 [KERBEROS] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," [RFC 1510](#), September 1993, <http://www.ietf.org/rfc/rfc1510.txt> .
- 1616
- 1617 [KEYWORDS] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels,"
- 1618 [RFC 2119](#), Harvard University, March 1997
- 1619 [SHA-1] FIPS PUB 180-1. Secure Hash Standard. U.S. Department of
- 1620 Commerce / National Institute of Standards and Technology.
- 1621 <http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>
- 1622 [SOAP11] W3C Note, "[SOAP: Simple Object Access Protocol 1.1](#)," 08 May 2000.
- 1623 [SOAP12] W3C [Recommendation](#), "[http://www.w3.org/TR/2003/REC-soap12-part1-](http://www.w3.org/TR/2003/REC-soap12-part1-20030624/)
- 1624 [20030624/](#)", 24 June 2003.
- 1625 [SOAPSEC] W3C Note, "[SOAP Security Extensions: Digital Signature](#)," 06 February
- 1626 2001.
- 1627 [URI] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers
- 1628 (URI): Generic Syntax," [RFC 2396](#), MIT/LCS, U.C. Irvine, Xerox
- 1629 Corporation, August 1998.
- 1630 [XPath] W3C Recommendation, "[XML Path Language](#)", 16 November 1999
- 1631 The following are non-normative references included for background and related material:
- 1632 [WS-SECURITY] "[Web Services Security Language](#)", IBM, Microsoft, VeriSign, April 2002.
- 1633 "[WS-Security Addendum](#)", IBM, Microsoft, VeriSign, August 2002.
- 1634 "[WS-Security XML Tokens](#)", IBM, Microsoft, VeriSign, August 2002.
- 1635 [XMLC14N] W3C Recommendation, "[Canonical XML Version 1.0](#)," 15 March 2001
- 1636 [EXCC14N] W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8
- 1637 July 2002.
- 1638 [XMLENC] W3C Working Draft, "[XML Encryption Syntax and Processing](#)," 04 March
- 1639 2002
- 1640 W3C Recommendation, "Decryption Transform for XML Signature", 10
- 1641 December 2002.
- 1642 [XML-ns] W3C Recommendation, "[Namespaces in XML](#)," 14 January 1999.
- 1643 [XMLSCHEMA] W3C Recommendation, "[XML Schema Part 1: Structures](#)," 2 May 2001.
- 1644 W3C Recommendation, "[XML Schema Part 2: Datatypes](#)," 2 May 2001.

Deleted: Working Draft

Deleted: SOAP Version 1.2 Part 1:
Messaging Framework

Deleted: 26

Deleted: 2

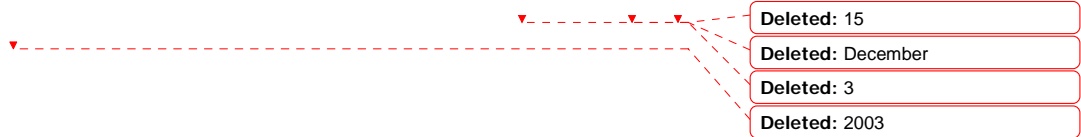
Deleted: 15

Deleted: December

Deleted: 3

Deleted: 2003

1645	[XMLSIG]	W3C Recommendation, " XML Signature Syntax and Processing ," 12 February 2002.
1646		
1647	[X509]	S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified Certificates Profile,"
1648		http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I
1649		
1650		
1651	[WSS-SAML]	OASIS Working Draft 06, " Web Services Security SAML Token Profile ", 21 February 2003
1652		
1653	[WSS-XrML]	OASIS Working Draft 03, " Web Services Security XrML Token Profile ", 30 January 2003
1654		
1655	[WSS-X509]	OASIS Working Draft 03, " Web Services Security X509 Profile ", 30 January 2003
1656		
1657	[WSSKERBEROS]	OASIS Working Draft 03, " Web Services Security Kerberos Profile ", 30 January 2003
1658		
1659	[WSSUSERNAME]	OASIS Working Draft 02, " Web Services Security UsernameToken Profile ", 23 February 2003
1660		
1661	[WSS-XCBF]	OASIS Working Draft 1.1, " Web Services Security XCBF Token Profile ", 30 March 2003
1662		
1663	[XPOINTER]	"XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation", DeRose, Maler, Daniel, 11 September 2001.
1664		



1665

Appendix A: Utility Elements and Attributes

1666
1667
1668
1669
1670

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

1671

A.1. Identification Attribute

1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688

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

This section is non-normative.

1689

A.2. Timestamp Elements

1690
1691
1692
1693
1694
1695
1696
1697
1698
1699

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

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

- Deleted: 15
- Deleted: December
- Deleted: 3
- Deleted: 2003

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

1700
1701
1702
1703
1704

A detailed description can be found in Section 10.

This section is non-normative.

A.3. General Schema Types

1705
1706
1707
1708
1709
1710
1711

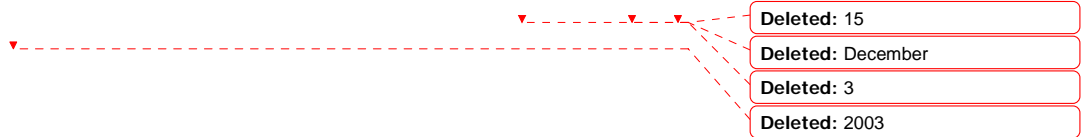
The schema for the utility aspects of this specification also defines some general purpose schema elements. While these elements are defined in this schema for use with this specification, they are general purpose definitions that may be used by other specifications as well.

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

1712
1713
1714

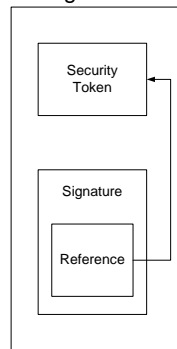
This section is non-normative.



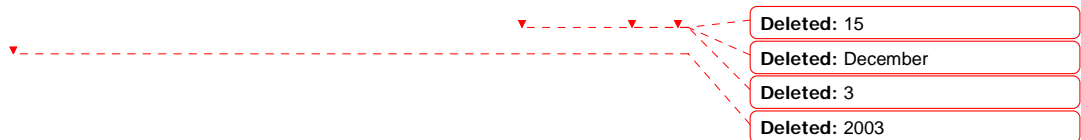
1715

Appendix B: SecurityTokenReference Model

1716 This appendix provides a non-normative overview of the usage and processing models for the
1717 `<wsse:SecurityTokenReference>` element.
1718 There are several motivations for introducing the `<wsse:SecurityTokenReference>`
1719 element:
1720 The XML Signature reference mechanisms are focused on "key" references rather than general
1721 token references.
1722 The XML Signature reference mechanisms utilize a fairly closed schema which limits the
1723 extensibility that can be applied.
1724 There are additional types of general reference mechanisms that are needed, but are not covered
1725 by XML Signature.
1726 There are scenarios where a reference may occur outside of an XML Signature and the XML
1727 Signature schema is not appropriate or desired.
1728 The XML Signature references may include aspects (e.g. transforms) that may not apply to all
1729 references.
1730
1731 The following use cases drive the above motivations:
1732 **Local Reference** – A security token, that is included in the message in the `<wsse:Security>`
1733 header, is associated with an XML Signature. The figure below illustrates this:



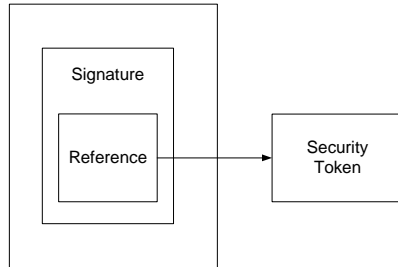
1734



1735
1736
1737

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:

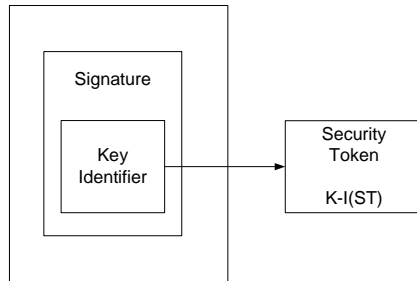
Deleted: <sp>
Deleted:



1738
1739
1740
1741

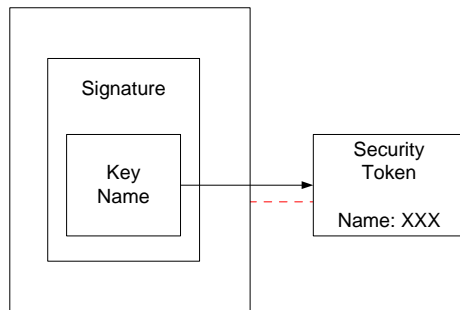
Key Identifier – A security token, which is associated with an XML Signature and identified using a known value that is the result of a well-known function of the security token (defined by the token format or profile). The figure below illustrates this where the token is located externally:

Deleted: ¶



1742
1743
1744
1745

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:

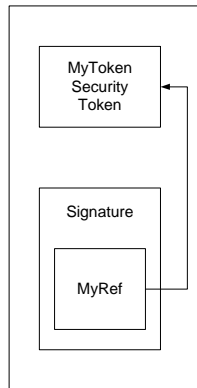


Deleted: 15
Deleted: December
Deleted: 3
Deleted: 2003

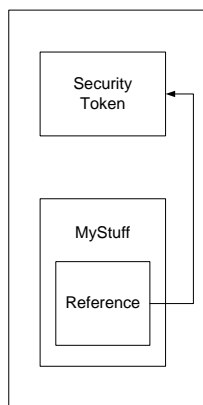
1746

1747 **Format-Specific References** – A security token is associated with an XML Signature and
 1748 identified using a mechanism specific to the token (rather than the general mechanisms
 1749 described above). The figure below illustrates this:

Deleted: <sp>



1750
 1751
 1752 | **Non-Signature References** – A message may contain XML that does not represent an XML
 1753 signature, but may reference a security token (which may or may not be included in the
 1754 message). The figure below illustrates this:



1755
 1756
 1757 | All conformant implementations **MUST** be able to process the
 1758 `<wss:SecurityTokenReference>` element. However, they are not required to support all of
 1759 the different types of references.
 1760 | The reference **MAY** include a *ValueType* attribute which provides a "hint" for the type of desired
 1761 token.
 1762 | If multiple sub-elements are specified, together they describe the reference for the token.
 WSS: SOAP Message Security
 Copyright © OASIS Open 2002-~~2003~~. All Rights Reserved.

Deleted: 15
 Deleted: December
 Deleted: 3
 Deleted: 2003

1763 | There are several challenges that implementations face when trying to interoperate:
 1764 | **ID References** – The underlying XML referencing mechanism using the XML base type of ID
 1765 | provides a simple straightforward XML element reference. However, because this is an XML
 1766 | type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
 1767 | requires the recipient to *understand* the schema. This may be an expensive task and in the
 1768 | general case impossible as there is no way to know the "schema location" for a specific
 1769 | namespace URI.

1770 | **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
 1771 | references are, by definition, unique by XML. However, other mechanisms such as "principal
 1772 | name" are not required to be unique and therefore such references may be unique.

1773 | The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
 1774 | information about the "key" used in the signature. For token references within signatures, it is
 1775 | RECOMMENDED that the `<wsse:SecurityTokenReference>` be placed within the
 1776 | `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
 1777 | by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
 1778 | Message Security or its profiles are preferred over the mechanisms in XML Signature.

1779 | The following provides additional details on the specific reference mechanisms defined in WSS:
 1780 | SOAP Message Security:

1781 | **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
 1782 | the security token. If only the fragment is specified, then it references the security token within
 1783 | the document whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to
 1784 | a [potentially external] security token identified using a URI. There are no implied semantics
 1785 | around the processing of the URI.

1786 | **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
 1787 | by specifying a known value (identifier) for the token, which is determined by applying a special
 1788 | *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
 1789 | specific security token but requires a profile or token-specific function to be specified. The
 1790 | `ValueType` attribute defines the type of key identifier and, consequently, identifies the type of
 1791 | token referenced. The `EncodingType` attribute specifies how the unique value (identifier) is
 1792 | encoded. For example, a hash value may be encoded using base 64 encoding (the default).

1793 | **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
 1794 | specific value that is used to *match* an identity assertion within the security token. This is a
 1795 | subset match and may result in multiple security tokens that match the specified name. While
 1796 | XML Signature doesn't imply formatting semantics, WSS: SOAP Message Security
 1797 | RECOMMENDS that X.509 names be specified.

1798 | It is expected that, where appropriate, profiles define if and how the reference mechanisms map
 1799 | to the specific token profile. Specifically, the profile should answer the following questions:
 1800 | What types of references can be used?
 1801 | How "Key Name" references map (if at all)?
 1802 | How "Key Identifier" references map (if at all)?
 1803 | Are there any additional profile or format-specific references?
 1804 |
 1805 | This section is non-normative.
 1806 |

Formatted: No bullets or numbering
 Deleted: 15
 Deleted: December
 Deleted: 3
 Deleted: 2003

1807

Appendix C: Revision History

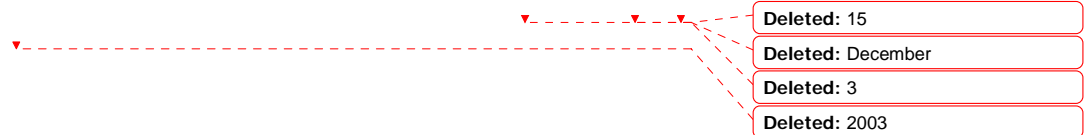
Rev	Date	What
01	20-Sep-02	Initial draft based on input documents and editorial review
02	24-Oct-02	Update with initial comments (technical and grammatical)
03	03-Nov-02	Feedback updates
04	17-Nov-02	Feedback updates
05	02-Dec-02	Feedback updates
06	08-Dec-02	Feedback updates
07	11-Dec-02	Updates from F2F
08	12-Dec-02	Updates from F2F
14	03-Jun-03	Completed these pending issues - 62, 69, 70, 72, 74, 84, 90, 94, 95, 96, 97, 98, 99, 101, 102, 103, 106, 107, 108, 110, 111
15	18-Jul-03	Completed these pending issues – 78, 82, 104, 105, 109, 111, 113
16	26-Aug-03	Completed these pending issues - 99, 128, 130, 132, 134
18	15-Dec-03	Editorial Updates based on Issue List #30
19	29-Dec	Editorial Updates based on Issue List #31

1808

1809

1810

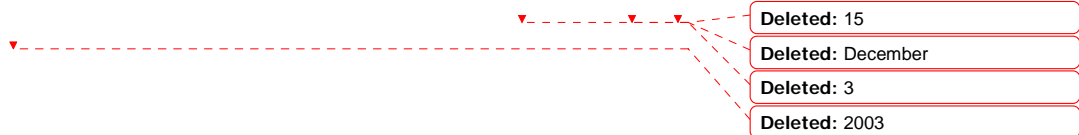
This section is non-normative.



1811

Appendix D: Notices

1812 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1813 that might be claimed to pertain to the implementation or use of the technology described in this
1814 document or the extent to which any license under such rights might or might not be available;
1815 neither does it represent that it has made any effort to identify any such rights. Information on
1816 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1817 website. Copies of claims of rights made available for publication and any assurances of licenses
1818 to be made available, or the result of an attempt made to obtain a general license or permission
1819 for the use of such proprietary rights by implementers or users of this specification, can be
1820 obtained from the OASIS Executive Director.
1821 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1822 applications, or other proprietary rights which may cover technology that may be required to
1823 implement this specification. Please address the information to the OASIS Executive Director.
1824 Copyright © OASIS Open 2002. *All Rights Reserved.*
1825 This document and translations of it may be copied and furnished to others, and derivative works
1826 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1827 published and distributed, in whole or in part, without restriction of any kind, provided that the
1828 above copyright notice and this paragraph are included on all such copies and derivative works.
1829 However, this document itself does not be modified in any way, such as by removing the
1830 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1831 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1832 Property Rights document must be followed, or as required to translate it into languages other
1833 than English.
1834 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1835 successors or assigns.
1836 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1837 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1838 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1839 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1840 PARTICULAR PURPOSE.
1841
1842 This section is non-normative.



Page 1: [1] Deleted Anthony Nadalin 1/4/2004 8:55 AM
15

Page 1: [1] Deleted Anthony Nadalin 1/4/2004 8:55 AM
December

Page 1: [1] Deleted Anthony Nadalin 1/4/2004 8:55 AM
3

Page 1: [2] Deleted Anthony Nadalin 1/4/2004 8:55 AM
15

Page 1: [2] Deleted Anthony Nadalin 1/4/2004 8:55 AM
December

Page 1: [2] Deleted Anthony Nadalin 1/4/2004 8:55 AM
3

Page 23: [3] Deleted Chris Kaler 1/5/2004 12:59 AM

QName	Description
TBD	TBD

Page 46: [4] Deleted Anthony Nadalin 1/5/2004 9:04 PM

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.

It is not feasible to provide a comprehensive list of security considerations for such a 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*.

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

There are many 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.

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

- Timestamp
- Sequence Number
- Expirations
- Message Correlation

This specification defines the use of XML Signature and XML Encryption in SOAP headers. As one of the building blocks for securing SOAP messages, it is intended to be used in conjunction with other security techniques. Digital signatures need to be understood in the context of other security mechanisms and possible threats to an entity. Digital signatures alone do not provide message authentication. One can record a signed message and resend it (a replay attack). To prevent this type of attack, digital signatures must be combined with an appropriate means to ensure the uniqueness of the message, such as timestamps or sequence numbers (see earlier section for additional details). The proper usage of nonce guards against replay attacks.

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

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

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

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

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

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

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

When a requester provides, within the request, a Public Key to be used to encrypt the response, it is possible that an attacker in the middle may substitute a different Public Key, thus allowing the attacker to read the response. The best way to prevent this attack is to bind the encryption key in some way to the request. One simple way of doing this is to use the same key pair to sign the request as to encrypt the response. However, if policy requires the use of distinct key pairs for signing and encryption, then the Public Key provided in the request should be included under the signature of the request.

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

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

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

When a password (or password equivalent) in a <UsernameToken> is used for authentication, the password needs to be properly protected. If the underlying transport does not provide enough protection against eavesdropping, the password SHOULD be digested as described in the WSS UsernameToken Profile specification. Even so, the password must be strong enough so that simple password guessing attacks will not reveal the secret from a captured message.

When a password is encrypted in addition to the normal threats against any encryption, two password-specific threats must be considered: replay and guessing. If an attacker can impersonate a user by replaying an encrypted or hashed password, then learning the actual password is not necessary. One method of preventing replay is to use a nonce as mentioned previously. Generally it is also necessary to use a timestamp to put a ceiling on the number of previous nonces that must be stored. However, in order to be effective the nonce and timestamp must be signed. If the signature is also over the password itself,

prior to encryption, then it would be a simple matter to use the signature to perform an offline guessing attack against the password. This threat can be countered in any of several ways including: don't include the password under the signature (the password will be verified later) or sign the encrypted password.

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

This section is non-normative