1 OASIS

Web Services Security:
SOAP Message Security

Working Draft 11, Monday, 03 March 2003

Document identifier:
> WSS: SOAP Message Security -11

Location:
> TBD

Editors:
> Phillip Hallam-Baker, VeriSign
> Chris Kaler, Microsoft
> Ronald Monzillo, Sun
> Anthony Nadalin, IBM

Contributors:

> TBD – Revise this list to include WSS TC contributors

Bob Atkinson, Microsoft	John Manferdelli, Microsoft
Giovanni Della-Libera, Microsoft	Hiroshi Maruyama, IBM
Satoshi Hada, IBM	Anthony Nadalin, IBM
Phillip Hallam-Baker, VeriSign	Nataraj Nagaratnam, IBM
Maryann Hondo, IBM	Hemma Prafullchandra, VeriSign
Chris Kaler, Microsoft	John Shewchuk, Microsoft
Johannes Klein, Microsoft	Dan Simon, Microsoft
Brian LaMacchia, Microsoft	Kent Tamura, IBM
Paul Leach, Microsoft	Hervey Wilson, Microsoft

Abstract:
> This specification describes enhancements to the SOAP messaging to provide quality of protection through message integrity, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.
>
> This specification also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required; it is designed to be extensible (e.g. support multiple security token formats). For example, a client might provide one format for proof of identity and provide another format for proof that they have a particular business certification.

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

30

31   **Status:**
32   This is an interim draft. Please send comments to the editors.

33

34   Committee members should send comments on this specification to the wss@lists.oasis-
35   open.org list. Others should subscribe to and send comments to the wss-
36   comment@lists.oasis-open.org list. To subscribe, visit http://lists.oasis-
37   open.org/ob/adm.pl.

38   For information on whether any patents have been disclosed that may be essential to
39   implementing this specification, and any offers of patent licensing terms, please refer to
40   the Intellectual Property Rights section of the Security Services TC web page
41   (http://www.oasis-open.org/who/intellectualproperty.shtml).

Table of Contents

113

114 1 Introduction

115 This specification proposes a standard set of SOAP extensions that can be used when building
116 secure Web services to implement message level integrity and confidentiality. This specification
117 refers to this set of extensions as the "Web Services Security Core Language" or "WSS-Core".

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

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

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

132 1.1 Goals and Requirements

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

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

138 As with every security protocol, significant efforts must be applied to ensure that security
139 protocols constructed using this specification are not vulnerable to any one of a wide range of
140 attacks.

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

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

145 1.1.1 Requirements

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

148 • Multiple security token formats

149 • Multiple trust domains

150 • Multiple signature formats

151 • Multiple encryption technologies

152 • End-to-end message-level security and not just transport-level security

1.1.2 Non-Goals

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

155 • Establishing a security context or authentication mechanisms.

156 • Key derivation.

157 • Advertisement and exchange of security policy.

158 • How trust is established or determined.

159

₁₆₀ 2 Notations and Terminology

₁₆₁ This section specifies the notations, namespaces, and terminology used in this specificatio n.

₁₆₂ 2.1 Notational Conventions

₁₆₃ The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
₁₆₄ "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
₁₆₅ document are to be interpreted as described in RFC 2119.

₁₆₆ When describing abstract data models, this specification uses the notational
₁₆₇ convention used by the XML Infoset. Specifically, abstract property names always
₁₆₈ appear in square brackets (e.g., [some property]).

₁₆₉ When describing concrete XML schemas, this specification uses the notational convent ion of
₁₇₀ WSS: SOAP Message Security . Specifically, each member of an element's [children] or
₁₇₁ [attributes] property is described using an XPath-like notation (e.g.,
₁₇₂ /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element
₁₇₃ w ildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute wildcard
₁₇₄ (<xs:anyAttribute/>)

₁₇₅ This specification is designed to work with the general SOAP message structure and message
₁₇₆ processing model, and should be applicable to any version of SOAP. The current SOAP 1.2
₁₇₇ namespace URI is used herein to provide detailed examples, but there is no intention to limit the
₁₇₈ applicability of this specification to a single version of SOAP.

₁₇₉ Readers are presumed to be familiar with the terms in the Internet Security Glossary.

₁₈₀ 2.2 Namespaces

₁₈₁ The XML namespace URIs that MUST be used by implementations of this specification are as
₁₈₂ follows (note that elements used in this specification are from various namespaces):

```
183          http://schemas.xmlsoap.org/ws/2002/xx/secext
184          http://schemas.xmlsoap.org/ws/2002/xx/utility
```

₁₈₅ The following namespaces are used in this document:

₁₈₆

Prefix	Namespace
S	http://www.w3.org/2002/12/soap-envelope
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#

wsse	http://schemas.xmlsoap.org/ws/2002/xx/secext
wsu	http://schemas.xmlsoap.org/ws/2002/xx/utility

2.3 Terminology

187

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

189 **Attachment** – An *attachment* is a generic term referring to additional data that travels with a
190 SOAP message, but is not part of the SOAP Envelope.

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

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

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

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

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

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

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

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

209 **Proof-of-Possession** – *Proof-of-possession* is authentication data that is provided with a
210 message to prove that the message was sent and or created by a claimed identity.

211 **Signature** - A *signature* is a value computed with a cryptographic algorithm and bound
212 to data in such a way that intended recipients of the data can use the signature to verify that the
213 data has not been altered since it was signed by the signer..

214 **Security Token** – A *security token* represents a collection (one or more) of claims.
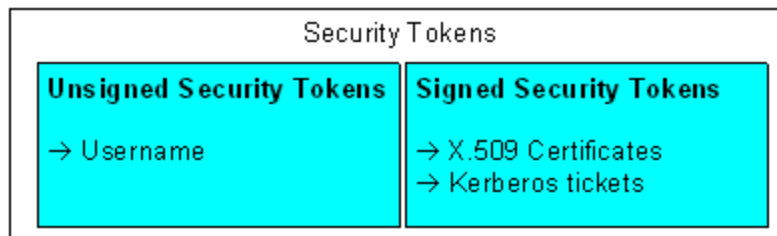
215

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

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

220 **Trust Domain** – A *Trust Domain* is a security space in which the target of a request can
221 determine whether particular sets of credentials from a source satisfy the relevant security
222 policies of the target. The target may defer trust to a third party thus including the trusted third
223 party in the Trust Domain.

224

225

226

227 3 Message Protection Mechanisms

228 When securing SOAP messages, various types of threats should be considered. This includes,
229 but is not limited to: 1) the message could be modified or read by antagonists or 2) an antagonist
230 could send messages to a service that, while well-formed, lack appropriate security claims to
231 warrant processing.

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

233 3.1 Message Security Model

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

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

244 Signatures are used to verify message origin and integrity. Signatures are also used by message
245 senders to demonstrate knowledge of the key used to confirm the claims in a security token and
246 thus to bind their identity (and any other claims occurring in the security token) to the messages
247 they create.

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

250 Where the specification requires that the elements be "processed" this means that the element
251 type be recognized well enough to return appropriate error if not supported.

252 3.2 Message Protection

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

257 Message integrity is provided by leveraging XML Signature in conjunction with security tokens to
258 ensure that messages are received without modifications. The integrity mechanisms are
259 designed to support multiple signatures, potentially by multiple SOAP roles, and to be extensible
260 to support additional signature formats.

261 Message confidentiality leverages XML Encryption in conjunction with security tokens to keep
262 portions of a SOAP message confidential. The encryption mechanisms are designed to support
263 additional encryption processes and operations by multiple SOAP roles.

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

265 This document also does not specify any signature appearing outside of `<wsse:Security>`

266 element, if any.

267 3.3 Invalid or Missing Claims

268 The message recipient SHOULD reject a message with a signature determined to be invalid,

269 missing or unacceptable claims as it is an unauthorized (or malformed) message. This

270 specification provides a flexible way for the message sender to make a claim about the security

271 properties by associating zero or more security tokens with the message. An example of a

272 security claim is the identity of the sender; the sender can claim that he is Bob, known as an

273 employee of some company, and therefore he has the right to send the message.

274 3.4 Example

275 The following example illustrates the use of a username security token containing a claimed

276 security identity to establish a password derived signing key. The password is not provided in the

277 security token. The message sender combines the password with the nonce and timestamp

278 appearing in the security token to define an HMAC signing key that it then uses to sign the

279 message. The message receiver uses its knowledge of the shared secret to repeat the HMAC

280 key calculation which it uses to validate the signature and in the process confirm that the

281 message was authored by the claimed user identity. The nonce and timestamp are used in the

282 key calculation to introduce variability in the keys derived from a given password value.

```
283  (001) <?xml version="1.0" encoding="utf-8"?>
284  (002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
285              xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
286  (003)   <S:Header>
287  (004)     <wsse:Security
288              xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
289  (005)       <wsse:UsernameToken wsu:Id="MyID">
290  (006)          <wsse:Username>Zoe</wsse:Username>
291  (007)          <wsse:Nonce>FKJh...</wsse:Nonce>
292  (008)          <wsu:Created>2001-10-13T09:00:00Z</wsu:Created>
293  (009)       </wsse:UsernameToken>
294  (010)       <ds:Signature>
295  (011)         <ds:SignedInfo>
296  (012)           <ds:CanonicalizationMethod
297                    Algorithm=
298                      "http://www.w3.org/2001/10/xml-exc-c14n#"/>
299  (013)           <ds:SignatureMethod
300                    Algorithm=
301                      "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
302  (014)           <ds:Reference URI="#MsgBody">
303  (015)             <ds:DigestMethod
304                      Algorithm=
305                        "http://www.w3.org/2000/09/xmldsig#sha1"/>
306  (016)             <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
307  (017)           </ds:Reference>
308  (018)         </ds:SignedInfo>
309  (019)         <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
310  (020)         <ds:KeyInfo>
311  (021)           <wsse:SecurityTokenReference>
```

```
312   (022)                        <wsse:Reference URI="#MyID"/>
313   (023)                     </wsse:SecurityTokenReference>
314   (024)               </ds:KeyInfo>
315   (025)            </ds:Signature>
316   (026)         </wsse:Security>
317   (027)    </S:Header>
318   (028)    <S:Body wsu:Id="MsgBody">
319   (029)       <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
320                   QQQ
321               </tru:StockSymbol>
322   (030)    </S:Body>
323   (031) </S:Envelope>
```

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

326 Line (004) starts the `<Security>` header defined in this specification. This header contains
327 security information for an intended recipient. This element continues until line (026)

328 Lines (005) to (009) specify a security token that is associated with the message. In this case, it
329 defines *username* of the client using the `<UsernameToken>`. Note that here the assumption is
330 that the service knows the password – in other words, it is a shared secret and the `<Nonce>` and
331 `<Created>` are used to generate the key

332 Lines (010) to (025) specify a digital signature. This signature ensures the integrity of the signed
333 elements. The signature uses the XML Signature specification identified by the ds namespace
334 declaration in Line (002). In this example, the signature is based on a key generated from the
335 user's password; typically stronger signing mechanisms would be used (see the Extended
336 Example later in this document).

337 Lines (011) to (018) describe what is being signed and the type of canonicalization being used.
338 Line (012) specifies how to canonicalize (normalize) the data that is being signed. Lines (014) to
339 (017) select the elements that are signed and how to digest them. Specifically, line (014)
340 indicates that the `<S:Body>` element is signed. In this example only the message body is
341 signed; typically all critical elements of the message are included in the signature (see the
342 Extended Example below).

343 Line (019) specifies the signature value of the canonicalized form of the data that is being signed
344 as defined in the XML Signature specification.

345 Lines (020) to (024) provide a *hint* as to where to find the security token associated with this
346 signature. Specifically, lines (021) to (023) indicate that the security token can be found at (pulled
347 from) the specified URL.

348 Lines (028) to (030) contain the *body* (payload) of the SOAP message.

349

350 4 ID References

351 There are many motivations for referencing other message elements such as signature
352 references or correlating signatures to security tokens. However, because arbitrary ID attributes
353 require the schemas to be available and processed, ID attributes which can be referenced in a
354 signature are restricted to the following list:

355 ID attributes from XML Signature

356 ID attributes from XML Encryption

357 wsu:Id global attribute described below

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

361 4.1 Id Attribute

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

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

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

376 4.2 Id Schema

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

380 The syntax for this attribute is as follows:

381
```
<anyElement wsu:Id="...">...</anyElement>
```

382 The following describes the attribute illustrated above:

383 *.../@wsu:Id*

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

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

387 Implementations MAY rely on XML Schema validation to provide rudimentary enforcement for

388　intra-document uniqueness. However, applications SHOULD NOT rely on schema validation
389　alone to enforce uniqueness.

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

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

```
393     <x:myElement wsu:Id="ID1" xmlns:x="..."
394                 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility"/>
```

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

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

405 5 Security Header

406 The `<wsse:Security>` header block provides a mechanism for attaching security-related
407 information targeted at a specific recipient in a form of a SOAP role. This MAY be either the
408 ultimate recipient of the message or an intermediary. Consequently, elements of this type MAY
409 be present multiple times in a SOAP message. An intermediary on the message path MAY add
410 one or more new sub-elements to an existing `<wsse:Security>` header block if they are
411 targeted for its SOAP node or it MAY add one or more new headers for additional targets.

412 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted
413 for separate recipients. However, only one `<wsse:Security>` header block MAY omit the
414 `S:role` attribute and no two `<wsse:Security>` header blocks MAy have the same value for
415 `S:role`. Message security information targeted for different recipients MUST appear in different
416 `<wsse:Security>` header blocks. The `<wsse:Security>` header block without a specified
417 `S:role` MAY be consumed by anyone, but MUST NOT be removed prior to the final destination
418 or endpoint.

419 As elements are added to the `<wsse:Security>` header block, they SHOULD be prepended to
420 the existing elements. As such, the `<wsse:Security>` header block represents the signing and
421 encryption steps the message sender took to create the message. This prepending rule ensures
422 that the receiving application MAY process sub-elements in the order they appear in the
423 `<wsse:Security>` header block, because there will be no forward dependency among the sub-
424 elements. Note that this specification does not impose any specific order of processing the sub-
425 elements. The receiving application can use whatever order is required.

426 When a sub-element refers to a key carried in another sub-element (for example, a signature
427 sub-element that refers to a binary security token sub-element that contains the X.509 certificate
428 used for the signature), the key-bearing security token SHOULD be prepended to the key-using
429 sub-element being added, so that the key material appears before the key-using sub-element.

430 The following illustrates the syntax of this header:

431
432
433
434
435
436
437
438
439
440
```
<S:Envelope>
    <S:Header>
            ...
        <wsse:Security S:role="..." S:mustUnderstand="...">
            ...
        </wsse:Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

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

442 */wsse:Security*

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

444 */wsse:Security/ @S:role*

445 This attribute allows a specific SOAP role to be identified. This attribute is optional;
446 however, no two instances of the header block may omit a role or specify the same role.

447   */wsse:Security/{any}*

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

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

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

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

454   All compliant implementations MUST declare which profiles they support and MUST be able to
455   process a `<wsse:Security>` element including any sub-elements which may be defined by that
456   profile.

457   The next few sections outline elements that are expected to be used within the
458   `<wsse:Security>` header.

459 6 Security Tokens

460 This chapter specifies some different types of security tokens and how they SHALL be attached
461 to messages.

462 6.1 Attaching Security Tokens

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

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

468 6.1.1 Processing Rules

469 This specification describes the processing rules for using and processing XML Signature and
470 XML Encryption. These rules MUST be followed when using any type of security token. Note
471 that this does NOT mean that security tokens MUST be signed or encrypted – only that if
472 signature or encryption is used in conjunction with security tokens, they MUST be used in a way
473 that conforms to the processing rules defined by this specification.

474 6.1.2 Subject Confirmation

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

478 6.2 User Name Token

479 6.2.1 Usernames

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

482 The following illustrates the syntax of this element:

```
<wsse:UsernameToken wsu:Id="...">
    <wsse:Username>...</wsse:Username>
</wsse:UsernameToken>
```
483
484
485

Formatte

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

487 */wsse:UsernameToken*

488   This element is used to represent a claimed identity.

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

490   A string label for this security token.

491 */wsse:UsernameToken/Username*

492   This required element specifies the claimed identity.

493 */wsse:UsernameToken/Username/@{any}*

494   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
495   the `<wsse:Username>` element.

496 /wsse:UsernameToken/{any}

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

499 */wsse:UsernameToken/@{any}*

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

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

503 The following illustrates the use of this:

```
504  <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
505              xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
506    <S:Header>
507          ...
508        <wsse:Security>
509            <wsse:UsernameToken>
510                <wsse:Username>Zoe</wsse:Username>
511            </wsse:UsernameToken>
512        </wsse:Security>
513          ...
514    </S:Header>
515    ...
516  </S:Envelope>
517
```

518 6.3 Binary Security Tokens

519 6.3.1 Attaching Security Tokens

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

523

524 6.3.2 Encoding Binary Security Tokens

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

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

532 The following is an overview of the syntax:

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

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

537 */wsse:BinarySecurityToken*

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

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

540   An optional string label for this security token.

541 */wsse:BinarySecurityToken/@ValueType*

542   The `ValueType` attribute is used to indicate the "value space" of the encoded binary
543   data (e.g. an X.509 certificate). The `ValueType` attribute allows a qualified name that
544   defines the value type and space of the encoded binary data. This attribute is extensible
545   using XML namespaces. Subsequent specifications MUST define the ValueType value
546   for the tokens that they define.

547 */wsse:BinarySecurityToken/@EncodingType*

548   The `EncodingType` attribute is used to indicate, using a QName, the encoding format of
549   the binary data (e.g., `wsse:Base64Binary`). A new attribute is introduced, as there
550   issues with the current schema validation tools that make derivations of mixed simple
551   and complex types difficult within XML Schema. The `EncodingType` attribute is
552   interpreted to indicate the encoding format of the element. The following encoding
553   formats are pre-defined:

QName	Description
wsse:Base64Binary	XML Schema base 64 encoding

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

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

557 All compliant implementations MUST be able to support a `<wsse:BinarySecurityToken>`
558 element.

559 When a `<wsse:BinarySecurityToken>` is included in a signature—that is, it is referenced
560 from a `<ds:Signature>` element—care should be taken so that the canonicalization algorithm
561 (e.g., Exclusive XML Canonicalization) does not allow unauthorized replacement of namespace
562 prefixes of the QNames used in the attribute or element values. In particular, it is
563 RECOMMENDED that these namespace prefixes be declared within the
564 `<wsse:BinarySecurityToken>` element if this token does not carry the validating key (and
565 consequently it is not cryptographically bound to the signature). For example, if we wanted to
566 sign the previous example, we need to include the consumed namespace definitions.

567 In the following example, a custom `ValueType` is used. Consequently, the namespace definition
568 for this `ValueType` is included in the `<wsse:BinarySecurityToken>` element. Note that the
569 definition of `wsse` is also included as it is used for the encoding type and the element.

570   `<wsse:BinarySecurityToken`

```
571              xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
572              wsu:Id="myToken"
573              ValueType="x:MyType" xmlns:x="http://www.fabrikam123.com/x"
574              EncodingType="wsse:Base64Binary">
575          MIIEZzCCA9CgAwIBAgIQEmtJZc0...
576      </wsse:BinarySecurityToken>
```

6.4 XML Tokens

This section presents the basic princip les and framework for using XML-based security tokens.
Subsequent specifications describe rules and processes for specific XML-based security token
formats.

6.4.1 Identifying and Referencing Security Tokens

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

590 7 Token References

591 This chapter discusses and defines mechanisms for referencing security tokens.

592 7.1 SecurityTokenReference Element

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

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

600 If a SecurityTokenRefeerence used outside of the `<Security>` header block the meaning of the
601 response and/or processing rules of the result ing reference are MUST be specified by the
602 containing element and are out of scope of this specification.

603 The following illustrates the syntax of this element:

```
604     <wsse:SecurityTokenReference wsu:Id="...">
605         ...
606     </wsse:SecurityTokenReference>
```

607 The following describes the elements defined above:

608 */wsse:SecurityTokenReference*

609   This element provides a reference to a security token.

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

611   A string label for this security token reference. This identifier names the reference. This
612   attribute does not indicate the ID of what is being referenced, that is done using a
613   fragment URI in a `<Reference>` element within the `<SecurityTokenReference>`
614   element.

615 */wsse:SecurityTokenReference/@wsse:Usage*

616   This optional attribute is used to type the usage of the `<SecurityToken>`. Usages are
617   specified using QNames and multiple usages MAY be specified using XML list
618   semantics.

QName	Description
TBD	TBD

619

620 */wsse:SecurityTokenReference/{any}*

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

623　*/wsse:SecurityTokenReference/@{any}*

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

626　All compliant implementations MUST be able to process a
627　`<wsse:SecurityTokenReference>` element.

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

633　There are several challenges that implementations face when trying to interoperate. In order to
634　process the IDs and references requires the recipient to *understand* the schema. This may be an
635　expensive task and in the general case impossible as there is no way to know the "schema
636　location" for a specific namespace URI. As well, **t**he primary goal of a reference is to uniquely
637　identify the desired token. ID references are, by definition, unique by XML. However, other
638　mechanisms such as "principal name" are not required to be unique and therefore such
639　references may be unique.

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

642　**Direct References** – This allows references to included tokens using URI fragments and external
643　tokens using full URIs.

644　**Key Identifiers** – This allows tokens to be referenced using an opaque value that represents the
645　token (defined by token type/profile).

646　**Key Names** – This allows tokens to bereferenced using a string that matches an identity
647　assertion within the security token. This is a subset match and may result in multiple security
648　tokens that match the specified name.

7.2 Direct References

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

652　The following illustrates the syntax of this element:

```
653    <wsse:SecurityTokenReference wsu:Id="...">
654       <wsse:Reference URI="..." ValueType="..."/>
655    </wsse:SecurityTokenReference>
```

656　The following describes the elements defined above:

657　*/wsse:SecurityTokenReference/Reference*

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

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

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

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

663　　　　This optional attribute specifies a QName that is used to identify the *type* of token being
664　　　　referenced (see `<wsse:BinarySecurityToken>`). This specification does not define

665   any processing rules around the usage of this attribute, however, specifications for
666   individual token types MAY define specific processing rules and semantics around the
667   value of the URI and how it SHALL be interpreted. If this attribute is not present, the URI
668   SHALL be processed as a normal URI.

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

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

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

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

675 The following illustrates the use of this element:

```
676    <wsse:SecurityTokenReference
677            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
678       <wsse:Reference
679            URI="http://www.fabrikam123.com/tokens/Zoe#X509token"/>
680    </wsse:SecurityTokenReference>
```

681 7.3 Key Identifiers

682 Alternatively, if a direct reference is not used, then it is RECOMMENDED to use a key identifier to
683 specify/reference a security token instead of a ds:KeyName. The `<wsse:KeyIdentifier>`
684 element SHALL be placed in the `<wsse:SecurityTokenReference>` element to reference a
685 token using an identifier. This element SHOULD be used for all key identifiers.

686 The processing model assumes that the key identifier for a security token is constant.
687 Consequently, processing a key identifier is simply looking for a security token whose key
688 identifier matches a given specified constant.

689 The following is an overview of the syntax:

```
690    <wsse:SecurityTokenReference>
691       <wsse:KeyIdentifier wsu:Id="..."
692                           ValueType="..."
693                           EncodingType="...">
694          ...
695       </wsse:KeyIdentifier>
696    </wsse:SecurityTokenReference>
```

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

698 */wsse: SecurityTokenReference /KeyIdentifier*

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

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

701   An optional string label for this identifier.

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

703   The `ValueType` attribute is used to optionally indicate the type of token with the
704   specified identifier. If specified, this is a *hint* to the recipient. Any value specified for
705   binary security tokens, or any XML token element QName can be specified here. If this
706   attribute isn't specified, then the identifier applies to any type of token.

707 */wsse: SecurityTokenReference/KeyIdentifier/ @EncodingType*

708	The optional `EncodingType` attribute is used to indicate, using a QName, the encoding
709	format of the binary data (e.g., `wsse:Base64Binary`). The base values defined in this
710	specification are used:

QName	Description
wsse:Base64Binary	XML Schema base 64 encoding (default)

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

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

714 **7.4 Embedded References**

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

718 The following is an overview of the syntax:

```
719     <wsse:SecurityTokenReference>
720       <wsse:Embedded wsu:Id="...">
721          ...
722       </wsse:Embedded>
723     </wsse:SecurityTokenReference>
```

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

725 */wsse: SecurityTokenReference /Embedded*

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

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

729   An optional string label for this element.

730 */wsse:SecurityTokenReference/KeyIdentifier/{any}*

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

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

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

736 The following example illustrates embedding a SAML assertion:

```
737     <S:Envelope>
738       <S:Header>
739         <wsse:Security>
740              ...
741           <wsse:SecurityTokenReference>
742               <wsse:Embedded wsu:Id="tok1">
743               </wsse:Embedded>
744           </wsse:SecurityTokenReference>
745              ...
```

```
746              <wsse:Security>
747          </S:Header>
748          ...
749      </S:Body>
```

7.5 ds:KeyInfo

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

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

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

7.6 Key Names

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

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

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

7.7 Token Reference Lookup Processing Order

768 There are a number of mechanisms described in XML Signature and this specification
769 for referencing security tokens. To resolve possible ambiguities when more than one
770 of these reference constructs is included in a single KeyInfo element, the following
771 processing order SHOULD be used:

772 1. Resolve any `<wsse:Reference>` elements (specified within
773   `<wsse:SecurityTokenReference>`).
774 2. Resolve any `<wsse:KeyIdentifier>` elements (specified within
775   `<wsse:SecurityTokenReference>`).
776 3. Resolve any `<ds:KeyName>` elements.
777 4. Resolve any other `<ds:KeyInfo>` elements.
778 The processing stops as soon as one key has been located.

779 8 Signatures

780 Message senders may want to enable message recipients to determine whether a message was
781 altered in transit and to verify that the claims in a particular security token apply to the sender of
782 the message.

783 Demonstrating knowledge of a confirmation key associated with a token key claim supports
784 confirming the other token claims. Knowledge of a confirmation key may be demonstrated using a
785 key to create an XML Signature, for example. The relying party acceptance of the claims may
786 depend on confidence in the token . Multiple tokens may have a key claim for a signature and
787 may be referenced from the signature using a SecurityTokenReference. A key claim can be an
788 X.509 Certificate token, or a Kerberos service ticket token to give two examples.

789 Because of the mutability of some SOAP headers, senders SHOULD NOT use the *Enveloped*
790 *Signature Transform* defined in XML Signature. Instead, messages SHOULD explicitly include
791 the elements to be signed. Similarly, senders SHOULD NOT use the *Enveloping Signature*
792 defined in XML Signature.

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

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

806 8.1 Algorithms

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

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

Algorithm Type	Algorithm	Algorithm URI
Canonicalization	Exclusive XML Canonicalization	http://www.w3.org/2001/10/xml-exc-c14n#
Transformations	XML Decryption Transformation	http://www.w3.org/2001/04/decrypt#

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

813 Finally, if a sender wishes to sign a message before encryption, they should use the Decryption
814 Transformation for XML Signature.

8.2 Signing Messages

816 The `<wsse:Security>` header block MAY be used to carry a signature compliant with the XML
817 Signature specification within a SOAP Envelope for the purpose of signing one or more elements
818 in the SOAP Envelope. Multiple signature entries MAY be added into a single SOAP Envelope
819 within the `<wsse:Security>` header block. Senders SHOULD take care to sign all important
820 elements of the message, but care MUST be taken in creating a signing policy that will not to sign
821 parts of the message that might legitimately be altered in transit.

822 SOAP applications MUST satisfy the following conditions:

823 The application MUST be capable of processing the required elements defined in the XML
824 Signature specification.

825 To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element
826 conforming to the XML Signature specification SHOULD be prepended to the existing content of
827 the `<wsse:Security>` header block. All the `<ds:Reference>` elements contained in the
828 signature SHOULD refer to a resource within the enclosing SOAP envelope, or in an attachment.

829 XPath filtering can be used to specify objects to be signed, as described in the XML Signature
830 specification. However, since the SOAP message exchange model allows intermediate
831 applications to modify the Envelope (add or delete a header block; for example), XPath filtering
832 does not always result in the same objects after message delivery. Care should be taken in using
833 XPath filtering so that there is no subsequent validation failure due to such modifications.

834 The problem of modification by intermediaries is applicable to more than just XPath processing.
835 Digital signatures, because of canonicalization and digests, present particularly fragile examples
836 of such relationships. If overall message processing is to remain robust, intermediaries must
837 exercise care that their transformations do not occur within the scope of a digitally signed
838 component.

839 Due to security concerns with namespaces, this specification strongly RECOMMENDS the use of
840 the "Exclusive XML Canonicalization" algorithm or another canonicalization algorithm that
841 provides equivalent or greater protection.

842 For processing efficiency it is RECOMMENDED to have the signature added and then the
843 security token pre-pended so that a processor can read and cache the token before it is used.

8.3 Signing Tokens

845 It is often desirable to sign security tokens that are included in a message or even external to the
846 message. The XML Signature specification provides several common ways for referencing
847 information to be signed such as URIs, IDs, and XPath, but some token formats may not allow
848 tokens to be referenced using URIs or IDs and XPaths may be undesirable in some situations.

849 This specification allows different tokens to have their own unique reference mechanisms which
850 are specified in their profile as extensions to the `<SecurityTokenReference>` element. This
851 element provides a uniform referencing mechanism that is guaranteed to work with all token

852 formats. Consequently, this specification defines a new ref erence option for XML Signature: the
853 STR Dereference Transform.

854 This transform is specified by the URI http://schemas.xmlsoap.org/2002/xx/STR- Transform and
855 when applied to a `<SecurityTokenReference>` element it means that the output is the token
856 referenced by the `<SecurityTokenReference>` element not the element itself .

857 The processing model is to echo the input to the transform except when a
858 `<SecurityTokenReference>` element is encountered. When one is found, the element is not
859 echoed, but instead, it is used to locate a token(s) matching the criteria and rules defines by the
860 `<SecurityTokenReference>` element and echo it (them) to the output. Consequently, the
861 output of the transformation is the resultant sequence representing the input with any
862 `<SecurityTokenReference>` elements replaced by the referenced security token(s) matched.

863 The following illustrates an example of this transformation which references a token contained
864 w ithin the message envelope:

```
865     ...
866     <wsse:SecurityTokenReference wsu:Id="Str1">
867          ...
868     </wsse:SecurityTokenReference>
869     ...
870     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
871         <SignedInfo>
872          ...
873           <Reference URI="#Str1">
874             <Transforms>
875               <ds:Transform
876                   Algorithm=
877                       "http://schemas.xmlsoap.org/2002/xx/STR-Transform"/>
878             </Transforms>
879             <DigestMethod Algorithm=
880                               "http://www.w3.org/2000/09/xmldsig#sha1"/>
881             <DigestValue>...</DigestValue>
882           </Reference>
883         </SignedInfo>
884         <SignatureValue></SignatureValue>
885     </Signature>
886     ...
```

8.4 Signature Validation

888 The validation of a `<ds:Signature>` element inside an `<wsse:Security>` header block
889 SHALL fail if

890   • the syntax of the content of the element does not conform to this specification, or

891   • the validation of the signature contained in the element fails according to the core
892   validation of the XML Signature specification, or

893   • the application applying its own validation policy rejects the message for some reason
894   (e.g., the signature is created by an untrusted key – verifying the previous two steps only
895   performs cryptographic validation of the signature).

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

898 8.5 Example

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

```
901    <?xml version="1.0" encoding="utf-8"?>
902    <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
903                xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
904                xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
905                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
906      <S:Header>
907        <wsse:Security>
908          <wsse:BinarySecurityToken
909                    ValueType="wsse:X509v3"
910                    EncodingType="wsse:Base64Binary"
911                    wsu:Id="X509Token">
912              MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
913          </wsse:BinarySecurityToken>
914          <ds:Signature>
915            <ds:SignedInfo>
916              <ds:CanonicalizationMethod Algorithm=
917                  "http://www.w3.org/2001/10/xml-exc-c14n#"/>
918              <ds:SignatureMethod Algorithm=
919                  "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
920              <ds:Reference URI="#myBody">
921                <ds:Transforms>
922                  <ds:Transform Algorithm=
923                      "http://www.w3.org/2001/10/xml-exc-c14n#"/>
924                </ds:Transforms>
925                <ds:DigestMethod Algorithm=
926                    "http://www.w3.org/2000/09/xmldsig#sha1"/>
927                <ds:DigestValue>EULddytSo1...</ds:DigestValue>
928              </ds:Reference>
929            </ds:SignedInfo>
930            <ds:SignatureValue>
931              BL8jdfToEb1l/vXcMZNNjPOV...
932            </ds:SignatureValue>
933            <ds:KeyInfo>
934                <wsse:SecurityTokenReference>
935                    <wsse:Reference URI="#X509Token"/>
936                </wsse:SecurityTokenReference>
937            </ds:KeyInfo>
938          </ds:Signature>
939        </wsse:Security>
940      </S:Header>
941      <S:Body wsu:Id="myBody">
942        <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
943          QQQ
944        </tru:StockSymbol>
945      </S:Body>
946    </S:Envelope>
```

⁹⁴⁷ # 9 Encryption

⁹⁴⁸ This specification allows encryption of any combination of body blocks, header blocks, any of
⁹⁴⁹ these sub-structures, and attachments by either a common symmetric key shared by the sender
⁹⁵⁰ and the recipient or a symmetric key carried in the message in an encrypted form.

⁹⁵¹ In order to allow this flexibility, this specification leverages the XML Encryption standard.
⁹⁵² Specifically what this specification describes is how three elements (listed below and defined in
⁹⁵³ XML Encryption) can be used within the `<wsse:Security>` header block. When a sender or
⁹⁵⁴ an intermediary encrypts portion(s) of a SOAP message using XML Encryption they MUST
⁹⁵⁵ prepend a sub-element to the `<wsse:Security>` header block. Furthermore, the encrypting
⁹⁵⁶ party MUST prepend the sub-element into the `<wsse:Security>` header block for the targeted
⁹⁵⁷ recipient that is expected to decrypt these encrypted portions. The combined process of
⁹⁵⁸ encrypting portion(s) of a message and adding one of these a sub-elements referring to the
⁹⁵⁹ encrypted portion(s) is called an encryption step hereafter. The sub-element should contain
⁹⁶⁰ enough information for the recipient to identify which portions of the message are to be decrypted
⁹⁶¹ by the recipient.

⁹⁶² All compliant implementations MUST be able to support the XML Encryption standard.

⁹⁶³ ## 9.1 xenc:ReferenceList

⁹⁶⁴ When encrypting elements or element contents within a SOAP envelope, the
⁹⁶⁵ `<xenc:ReferenceList>` element from XML Encryption MAY be used to create a manifest of
⁹⁶⁶ encrypted portion(s), which are expressed as `<xenc:EncryptedData>` elements within the
⁹⁶⁷ envelope. An element or element content to be encrypted by this encryption step MUST be
⁹⁶⁸ replaced by a corresponding `<xenc:EncryptedData>` according to XML Encryption. All the
⁹⁶⁹ `<xenc:EncryptedData>` elements created by this encryption step SHOULD be listed in
⁹⁷⁰ `<xenc:DataReference>` elements inside an `<xenc:ReferenceList>` element.

⁹⁷¹ Although in XML Encryption, `<xenc:ReferenceList>` is originally designed to be used within
⁹⁷² an `<xenc:EncryptedKey>` element (which implies that all the referenced
⁹⁷³ `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows
⁹⁷⁴ that `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>`
⁹⁷⁵ MAY be encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>`
⁹⁷⁶ within individual `<xenc:EncryptedData>`.

⁹⁷⁷ A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the sender
⁹⁷⁸ and the recipient use a shared secret key. The following illustrates the use of this sub-element:

```
979    <S:Envelope
980       xmlns:S="http://www.w3.org/2001/12/soap-envelope"
981       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
982       xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
983       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
984        <S:Header>
985            <wsse:Security>
986                <xenc:ReferenceList>
987                    <xenc:DataReference URI="#bodyID"/>
988                </xenc:ReferenceList>
```

```
989            </wsse:Security>
990         </S:Header>
991         <S:Body>
992            <xenc:EncryptedData Id="bodyID">
993              <ds:KeyInfo>
994                <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
995              </ds:KeyInfo>
996              <xenc:CipherData>
997                <xenc:CipherValue>...</xenc:CipherValue>
998              </xenc:CipherData>
999            </xenc:EncryptedData>
1000        </S:Body>
1001      </S:Envelope>
```

9.2 xenc:EncryptedKey

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

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

```
1014      <S:Envelope
1015         xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1016         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1017         xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
1018         xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1019        <S:Header>
1020            <wsse:Security>
1021                <xenc:EncryptedKey>
1022                    <xenc:EncryptionMethod Algorithm="..."/>
1023                    <ds:KeyInfo>
1024                       <wsse:SecurityTokenReference>
1025                       <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1026                            ValueType="wsse:X509v3">MIGfMa0GCSq...
1027                       </wsse:KeyIdentifier>
1028                        </wsse:SecurityTokenReference>
1029                    </ds:KeyInfo>
1030                    <xenc:CipherData>
1031                        <xenc:CipherValue>...</xenc:CipherValue>
1032                    </xenc:CipherData>
1033                    <xenc:ReferenceList>
1034                        <xenc:DataReference URI="#bodyID"/>
1035                    </xenc:ReferenceList>
1036                </xenc:EncryptedKey>
1037            </wsse:Security>
1038        </S:Header>
1039        <S:Body>
```

```
1040            <xenc:EncryptedData Id="bodyID">
1041                <xenc:CipherData>
1042                  <xenc:CipherValue>...</xenc:CipherValue>
1043                </xenc:CipherData>
1044            </xenc:EncryptedData>
1045        </S:Body>
1046    </S:Envelope>
```

1047 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in
1048 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that
1049 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

9.3 xenc:EncryptedData

1051 In some cases security-related information is provided in a purely encrypted form or non-XML
1052 attachments MAY be encrypted. The `<xenc:EncryptedData>` element from XML Encryption
1053 SHALL be used for these scenarios. For each part of the encrypted attachment, one encryption
1054 step is needed; that is, for each attachment to be encrypted, one `<xenc:EncryptedData>` sub-
1055 element MUST be added with the following rules (note that steps 2-4 applies only if MIME types
1056 are being used for attachments).

1057 The contents of the attachment MUST be replaced by the encrypted octet string.

1058 The replaced MIME part MUST have the media type `application/octet-stream`.

1059 The original media type of the attachment MUST be declared in the `MimeType` attribute of the
1060 `<xenc:EncryptedData>` element.

1061 The encrypted MIME part MUST be referenced by an `<xenc:CipherReference>` element with
1062 a URI that points to the MIME part with `cid:` as the scheme component of the URI.

1063 The following illustrates the use of this element to indicate an encrypted attachment:

```
1064    <S:Envelope
1065       xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1066       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1067       xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
1068       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1069        <S:Header>
1070            <wsse:Security>
1071                <xenc:EncryptedData MimeType="image/png">
1072                <ds:KeyInfo>
1073                    <wsse:SecurityTokenReference>
1074                  <xenc:EncryptionMethod Algorithm="..."/>
1075                  <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1076                        ValueType="wsse:X509v3">MIGfMa0GCSq...
1077                  </wsse:KeyIdentifier>
1078                    </wsse:SecurityTokenReference>
1079                </ds:KeyInfo>
1080                <xenc:CipherData>
1081                    <xenc:CipherReference URI="cid:image"/>
1082                </xenc:CipherData>
1083                </xenc:EncryptedData>
1084            </wsse:Security>
1085        </S:Header>
1086        <S:Body> </S:Body>
1087    </S:Envelope>
```

9.4 Processing Rules

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

When an element or element content inside a SOAP envelope (e.g. of the contents of `<S:Body>`) is to be encrypted, it MUST be replaced by an `<xenc:EncryptedData>`, according to XML Encryption and it SHOULD be referenced from the `<xenc:ReferenceList>` element created by this encryption step. This specification allows placing the encrypted octet stream in an attachment. For example, if an `<xenc:EncryptedData>` element in an `<S:Body>` element has `<xenc:CipherReference>` that refers to an attachment, then the decrypted octet stream SHALL replace the `<xenc:EncryptedData>`. However, if the `<enc:EncryptedData>` element is located in the `<Security>` header block and it refers to an attachment, then the decrypted octet stream MUST replace the encrypted octet stream in the attachment.

9.4.1 Encryption

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

Create a new SOAP envelope.

Create a `<Security>` header

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.

Locate data items to be encrypted, i.e., XML elements, element contents within the target SOAP envelope, and attachments.

Encrypt the data items as follows: For each XML element or element content within the target SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification. Each selected original element or element content MUST be removed and replaced by the resulting `<xenc:EncryptedData>` element. For an attachment, the contents MUST be replaced by encrypted cipher data as described in section 9.3 Signature Validation.

The optional `<ds:KeyInfo>` element in the `<xenc:EncryptedData>` element MAY reference another `<ds:KeyInfo>` element. Note that if the encryption is based on an attached security token, then a `<SecurityTokenReference>` element SHOULD be added to the `<ds:KeyInfo>` element to facilitate locating it.

Create an `<xenc:DataReference>` element referencing the generated `<xenc:EncryptedData>` elements. Add the created `<xenc:DataReference>` element to the `<xenc:ReferenceList>`.

9.4.2 Decryption

On receiving a SOAP envelope containing encryption header elements, for each encryption header element the following general steps should be processed (non-normative):

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

Decrypt them as follows: For each element in the target SOAP envelope, decrypt it according to the processing rules of the XML Encryption specification and the processing rules listed above.

If the decrypted data is part of an attachment and MIME types were used, then revise the MIME type of the attachment to the original MIME type (if one exists).

If the decryption fails for some reason, applications MAY report the failure to the sender using the fault code defined in Section 12 Error Handling.

9.5 Decryption Transformation

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

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

10 Message Timestamps

It is often important for the recipient to be able to determine the *freshness* of a message. In some cases, a message may be so *stale* that the recipient may decide to ignore it.

This specification does not provide a mechanism for synchronizing time. The assumption is either that the recipient is using a mechanism to synchronize time (e.g. NTP) or, more likely for federated applications, that they are making assessments about time based on three factors: creation time of the message, transmission checkpoints, and transmission delays and their local time.

To assist a recipient in making an assessment of staleness, a requestor may wish to indicate a suggested expiration time after which the recipient should ignore the message. The specification provides XML elements by which the requestor may express the expiration time of a message, the requestor's clock time at the moment the message was created, checkpoint timestamps (when an SOAP role received the message) along the communication path, and the delays introduced by transmission and other factors subsequent to creation. The quality of the delays is a function of how well they reflect the actual delays (e.g., how well they reflect transmission delays).

It should be noted that this is not a protocol for making assertions or determining when, or how fast, a service produced or processed a message.

This specification defines and illustrates time references in terms of the *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this type. It is further RECOMMENDED that all references be in UTC time. If, however, other time types are used, then the *ValueType* attribute (described below) MUST be specified to indicate the data type of the time format. Requestors and receivers SHOULD NOT rely on other applications supporting time resolution finer than milliseconds. Implementations MUST NOT generate time instants that specify leap seconds.

10.1 Model

This specification provides several tools for recipients to process the expiration time presented by the requestor. The first is the creation time. Recipients can use this value to assess possible clock skew . However, to make some assessments, the time required to go from the requestor to the recipient may also be useful in making this assessment. Two mechanisms are provided for this. The first is that intermediaries may add timestamp elements indicating when they received the message. This knowledge can be useful to get a holistic view of clocks along the message path. The second is that intermediaries can specify any delays they imposed on message delivery. It should be noted that not all delays can be accounted for, such as wire time and parties that don't report. Recipients need to take this into account when evaluating clock skew .

10.2 Timestamp Elements

This specification defines the following message timestamp elements. These elements are defined for use with the `<wsu:Timestamp>` header for SOAP messages, but they can be used anywhere within the header or body that creation, expiration, and delay times are needed.

10.2.1 Creation

1188

1189 The `<wsu:Created>` element specifies a creation timestamp. The exact meaning and
1190 semantics are dependent on the context in which the element is used. The syntax f or this
1191 element is as follows:

1192   `<wsu:Created ValueType="..." wsu:Id="...">...</wsu:Created>`

1193 The following describes the attributes and elements listed in the schema above:

1194 */wsu:Created*

1195   This element's value is a creation timestamp. Its type is specified by the ValueType
1196   attribute.

1197 */wsu:Created/@ValueType*

1198   This optional attribute specifies the type of the time data. This is specified as the XML
1199   Schema type. The default value is `xsd:dateTime`.

1200 */wsu:Created/@wsu:Id*

1201   This optional attribute specifies an XML Schema ID that can be used to reference this
1202   element.

10.2.2 Expiration

1203

1204 The `<wsu:Expires>` element specifies the expiration time. The exact meaning and processing
1205 rules for expiration depend on the context in which the element is used. The syntax for this
1206 element is as follows:

1207   `<wsu:Expires  ValueType="..." wsu:Id="...">...</wsu:Expires>`

1208 The following describes the attributes and elements listed in the schema above:

1209 */wsu:Expires*

1210   This element's value represents an expiration time. Its type is specified by the ValueType
1211   attribute

1212 */wsu:Expires/@ValueType*

1213   This optional attribute specifies the type of the time data. This is specified as the XML
1214   Schema type. The default value is `xsd:dateTime`.

1215 */wsu:Expires/@wsu:Id*

1216   This optional attribute specifies an XML Schema ID that can be used to reference this
1217   element.

1218 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
1219 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
1220 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
1221 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
1222 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
1223 judgment of the requestor's likely current clock time by means not described in this specification,
1224 for example an out-of-band clock synchronization protocol. The recipient may also use the
1225 creation time and the delays introduced by intermediate SOAP roles to estimate the degree of
1226 clock skew .

1227 One suggested formula for estimating clock skew is

```
1228        skew = recipient's arrival time - creation time - transmission time
```
1229 Transmission time may be estimated by summing the values of delay elements, if present. It
1230 should be noted that wire-time is only part of this if delays include it in estimates. Otherwise the
1231 transmission time will not reflect the on-wire time. If no delays are present, there are no special
1232 assumptions that need to be made about processing time

10.3 Timestamp Header

1234 A `<wsu:Timestamp>` header provides a mechanism for expressing the creation and expiration
1235 times of a message introduced throughout the message path. Specifically, is uses the previously
1236 defined elements in the context of message creation, receipt, and processing.

1237 All times SHOULD be in UTC format as specified by the XML Schema type (dateTime). It should
1238 be noted that times support time precision as defined in the XML Schema specification.

1239 Multiple `<wsu:Timestamp>` headers can be specified if they are targeted at different SOAP
1240 roles. The ordering within the header is as illustrated below.

1241 The ordering of elements in this header is fixed and MUST be preserved by intermediaries.

1242 To preserve overall integrity of each `<wsu:Timestamp>` header, it is strongly RECOMMENDED
1243 that each SOAP role create or update the appropriate `<wsu:Timestamp>` header destined to
1244 itself.

1245 The schema outline for the `<wsu:Timestamp>` header is as follows:

```
1246        <wsu:Timestamp wsu:Id="...">
1247            <wsu:Created>...</wsu:Created>
1248            <wsu:Expires>...</wsu:Expires>
1249            ...
1250        </wsu:Timestamp>
```

1251 The following describes the attributes and elements listed in the schema above:

1252 */wsu:Timestamp*

1253   This is the header for indicating message timestamps.

1254 */wsu:Timestamp/Created*

1255   This represents the creation time of the message. This element is optional, but can only
1256   be specified once in a `Timestamp` header. Within the SOAP processing model, creation
1257   is the instant that the infoset is serialized for transmission. The creation time of the
1258   message SHOULD NOT differ substantially from its transmission time. The difference in
1259   time should be minimized.

1260 */wsu:Timestamp/Expires*

1261   This represents the expiration of the message. This is optional, but can appear at most
1262   once in a `Timestamp` header. Upon expiration, the requestor asserts that the message
1263   is no longer valid. It is strongly RECOMMENDED that recipients (anyone who processes
1264   this message) discard (ignore) any message that has passed its expiration. A Fault code
1265   (wsu:MessageExpired) is provided if the recipient wants to inform the requestor that its
1266   message was expired. A service MAY issue a Fault indicating the message has expired.

1267 */wsu:Timestamp/{any}*

1268   This is an extensibility mechanism to allow additional elements to be added to the
1269   header.

1270   */wsu:Timestamp/@wsu:Id*

1271   This optional attribute specifies an XML Schema ID that can be used to reference this
1272   element.

1273   */wsu:Timestamp/@{any}*

1274   This is an extensibility mechanism to allow additional attributes to be added to the
1275   header.

1276   The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```
1277    <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1278                xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
1279      <S:Header>
1280        <wsu:Timestamp>
1281           <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
1282           <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
1283        </wsu:Timestamp>
1284        ...
1285      </S:Header>
1286      <S:Body>
1287        ...
1288      </S:Body>
1289    </S:Envelope>
```

1290 10.4 TimestampTrace Header

1291   A `<wsu:TimestampTrace>` header provides a mechanism for expressing the delays introduced
1292   throughout the message path. Specifically, is uses the previously defined elements in the context
1293   of message creation, receipt, and processing.

1294   All times SHOULD be in UTC format as specified by the XML Schema type (dateTime). It should
1295   be noted that times support time precision as defined in the XML Schema specification.

1296   Multiple `<wsu:TimestampTrace>` headers can be specified if they reference a different SOAP
1297   role.

1298   The `<wsu:Received>` element specifies a receipt timestamp with an optional processing delay.
1299   The exact meaning and semantics are dependent on the context in which the element is used.

1300   It is also strongly RECOMMENDED that each SOAP role sign its elements by referencing their
1301   ID, NOT by signing the `TimestampTrace` header as the header is mutable.

1302   The syntax for this element is as follows:

```
1303    <wsu:TimestampTrace>
1304      <wsu:Received Role="..." Delay="..." ValueType="..."
1305                wsu:Id="...">...</wsu:Received>
1306    </wsu:TimestampTrace>
```

1307   The following describes the attributes and elements listed in the schema above:

1308   */wsu:Received*

1309   This element's value is a receipt timestamp. The time specified SHOULD be a UTC
1310   format as specified by the ValueType attribute (default is XML Schema type dateTime).

1311   */wsu:Received/@Role*

1312	A required attribute, `Role`, indicates which SOAP role is indicating receipt. Roles MUST
1313	include this attribute, with a value matching the role value as specified as a SOAP
1314	intermediary.

1315 */wsu:Received/@Delay*

1316 The value of this optional attribute is the delay associated with the SOAP role expressed
1317 in milliseconds. The delay represents processing time by the Role after it received the
1318 message, but before it forwarded to the next recipient.

1319 */wsu:Received/@ValueType*

1320 This optional attribute specifies the type of the time data (the element value). This is
1321 specified as the XML Schema type. If this attribute isn't specified, the default value is
1322 `xsd:dateTime`.

1323 */wsu:Received/@wsu:Id*

1324 This optional attribute specifies an XML Schema ID that can be used to reference this
1325 element.

1326 The delay attribute indicates the time delay attributable to an SOAP role (intermediate
1327 processor). In some cases this isn't known; for others it can be computed as *role's send time –*
1328 *role's receipt time*.

1329 Each delay amount is indicated in units of milliseconds, without fractions. If a delay amount
1330 would exceed the maximum value expressible in the datatype, the value should be set to the
1331 maximum value of the datatype.

1332 The following example illustrates the use of the `<wsu:Timestamp>` header and a
1333 `<wsu:TimestampTrace>` header indicating a processing delay of one minute subsequent to the
1334 receipt which was two minutes after creation.

```
1335   <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1336               xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
1337     <S:Header>
1338       <wsu:Timestamp>
1339         <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
1340         <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
1341       </wsu:Timestamp>
1342       <wsu:TimespampTrace>
1343         <wsu:Received Role="http://x.com/" Delay="60000">
1344                 2001-09-13T08:44:00Z</wsu:Received>
1345       </wsu:TimestampTrace>
1346       ...
1347     </S:Header>
1348     <S:Body>
1349       ...
1350     </S:Body>
1351   </S:Envelope>
1352
```

1353 11 Extended Example

1354 The following sample message illustrates the use of security tokens, signatures, and encryption.
1355 For this example, the timestamp and the message body are signed prior to encryption. The
1356 decryption transformation is not needed as the signing/encryption order is specified within the
1357 `<wsse:Security>` header.

```
1358    (001) <?xml version="1.0" encoding="utf-8"?>
1359    (002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1360                 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1361                 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
1362                 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility"
1363                 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1364    (003)   <S:Header>
1365    (004)       <wsu:Timestamp>
1366    (005)           <wsu:Created wsu:Id="T0">
1367    (006)                2001-09-13T08:42:00Z
1368    (007)           </wsu:Created>
1369    (008)       </wsu:Timestamp>
1370    (009)       <wsse:Security>
1371    (010)           <wsse:BinarySecurityToken
1372                         ValueType="wsse:X509v3"
1373                         wsu:Id="X509Token"
1374                         EncodingType="wsse:Base64Binary">
1375    (011)          MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1376    (012)           </wsse:BinarySecurityToken>
1377    (013)           <xenc:EncryptedKey>
1378    (014)               <xenc:EncryptionMethod Algorithm=
1379                            "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
1380    (015)               <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1381    (016)                   ValueType="wsse:X509v3">MIGfMa0GCSq...
1382    (017)               </wsse:KeyIdentifier>
1383    (018)               <xenc:CipherData>
1384    (019)                   <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1385    (020)                   </xenc:CipherValue>
1386    (021)               </xenc:CipherData>
1387    (022)               <xenc:ReferenceList>
1388    (023)                    <xenc:DataReference URI="#enc1"/>
1389    (024)               </xenc:ReferenceList>
1390    (025)           </xenc:EncryptedKey>
1391    (026)           <ds:Signature>
1392    (027)               <ds:SignedInfo>
1393    (028)                   <ds:CanonicalizationMethod
1394                        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1395    (029)                   <ds:SignatureMethod
1396                       Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
1397    (039)                   <ds:Reference URI="#T0">
1398    (031)                       <ds:Transforms>
1399    (032)                           <ds:Transform
1400                        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1401    (033)                       </ds:Transforms>
```

```
1402    (034)                        <ds:DigestMethod
1403                          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
1404    (035)                        <ds:DigestValue>LyLsF094hPi4wPU...
1405    (036)                          </ds:DigestValue>
1406    (037)                     </ds:Reference>
1407    (038)                     <ds:Reference URI="#body">
1408    (039)                        <ds:Transforms>
1409    (040)                           <ds:Transform
1410                          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1411    (041)                        </ds:Transforms>
1412    (042)                        <ds:DigestMethod
1413                          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
1414    (043)                        <ds:DigestValue>LyLsF094hPi4wPU...
1415    (044)                          </ds:DigestValue>
1416    (045)                     </ds:Reference>
1417    (046)                  </ds:SignedInfo>
1418    (047)                  <ds:SignatureValue>
1419    (048)                          Hp1ZkmFZ/2kQLXDJbchm5gK...
1420    (049)                  </ds:SignatureValue>
1421    (050)                  <ds:KeyInfo>
1422    (051)                     <wsse:SecurityTokenReference>
1423    (052)                        <wsse:Reference URI="#X509Token"/>
1424    (053)                     </wsse:SecurityTokenReference>
1425    (054)                  </ds:KeyInfo>
1426    (055)              </ds:Signature>
1427    (056)         </wsse:Security>
1428    (057)    </S:Header>
1429    (058)    <S:Body wsu:Id="body">
1430    (059)        <xenc:EncryptedData
1431                        Type="http://www.w3.org/2001/04/xmlenc#Element"
1432                        wsu:Id="enc1">
1433    (060)          <xenc:EncryptionMethod
1434                        Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
1435    (061)          <xenc:CipherData>
1436    (062)             <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1437    (063)             </xenc:CipherValue>
1438    (064)          </xenc:CipherData>
1439    (065)        </xenc:EncryptedData>
1440    (066)    </S:Body>
1441    (067) </S:Envelope>
```

1442   Let's review some of the key sections of this example:

1443   Lines (003)-(057) contain the SOAP message headers.

1444   Lines (004)-(008) specify the timestamp information. In this case it indicates the creation time of
1445   the message.

1446   Lines (009)-(056) represent the `<wsse:Security>` header block. This contains the security-
1447   related information for the message.

1448   Lines (010)-(012) specify a security token that is associated with the message. In this case, it
1449   specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64
1450   encoding of the certificate.

1451   Lines (013)-(025) specify the key that is used to encrypt the body of the message. Since this is a
1452   symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
1453   encrypt the key. Lines (015)-(017) specify the name of the key that was used to encrypt the

1454   symmetric key. Lines (018)-(021) specify the actual encrypted form of the symmetric key. Lines
1455   (022)-(024) identify the encryption block in the message that uses this symmetric key. In this
1456   case it is only used to encrypt the body (Id="enc1").

1457   Lines (026)-(055) specify the digital signature. In this example, the signature is based on the
1458   X.509 certificate. Lines (027)-(046) indicate what is being signed. Specifically, Line (039)
1459   references the creation timestamp and line (038) references the message body.

1460   Lines (047)-(049) indicate the actual signature value – specified in Line (042).

1461   Lines (051)-(053) indicate the key that was used for the signature. In this case, it is the X.509
1462   certificate included in the message. Line (052) provides a URI link to the Lines (010)-(012).

1463   The body of the message is represented by Lines (056)-(066).

1464   Lines (059)-(065) represent the encrypted metadata and form of the body using XML Encryption.
1465   Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line
1466   (060) specifies the encryption algorithm – Triple-DES in this case. Lines (062)-(063) contain the
1467   actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
1468   key as the key references this encryption – Line (023).

₁₄₆₉ 12 Error Handling

1470 There are many circumstances where an *error* can occur while processing security information.
1471 For example:

1472 Invalid or unsupported type of security token, signing, or encryption

1473 Invalid or unauthenticated or unauthenticatable security token

1474 Invalid signature

1475 Decryption failure

1476 Referenced security token is unavailable

1477 Unsupported namespace

1478 These can be grouped into two *classes* of errors: unsupported and failure. For the case of
1479 unsupported errors, the recipient MAY provide a response that informs the sender of supported
1480 formats, etc. For failure errors, the recipient MAY choose not to respond, as this may be a form
1481 of Denial of Service (DOS) or cryptographic attack. We combine signature and encryption
1482 failures to mitigate certain types of attacks.

1483 If a failure is returned to a sender then the failure MUST be reported using SOAPs Fault
1484 mechanism. The following tables outline the predefined security fault codes. The "unsupported"
1485 class of errors are:

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

1486 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

Referenced security token could not be retrieved	wsse:SecurityTokenUnavailable

13 Security Considerations

¹⁴⁸⁷

¹⁴⁸⁸ It is strongly RECOMMENDED that messages include digitally signed elements to allow message
¹⁴⁸⁹ recipients to detect replays of the message when the messages are exchanged via an open
¹⁴⁹⁰ network. These can be part of the message or of the headers defined from other SOAP
¹⁴⁹¹ extensions. Four typical approaches are:

¹⁴⁹² Timestamp

¹⁴⁹³ Sequence Number

¹⁴⁹⁴ Expirations

¹⁴⁹⁵ Message Correlation

¹⁴⁹⁶ This specification defines the use of XML Signature and XML Encryption in SOAP headers. As
¹⁴⁹⁷ one of the building blocks for securing SOAP messages, it is intended to be used in conjunction
¹⁴⁹⁸ w ith other security techniques. Digital signatures need to be understood in the context of other
¹⁴⁹⁹ security mechanisms and possible threats to an entity.

¹⁵⁰⁰ Digital signatures alone do not provide message authentication. One can record a signed
¹⁵⁰¹ message and resend it (a replay attack). To prevent this type of attack, digital signatures must be
¹⁵⁰² combined with an appropriate means to ensure the uniqueness of the message, such as
¹⁵⁰³ timestamps or sequence numbers (see earlier section for additional details). The proper usage of
¹⁵⁰⁴ nonce guards aginst replay attacts.

¹⁵⁰⁵ When digital signatures are used for verifying the claims pertain ing to the sending entity, the
¹⁵⁰⁶ sender must demonstrate knowledge of the confirmation key. One way to achieve this is to use a
¹⁵⁰⁷ challenge-response type of protocol. Such a protocol is outside the scope of this document.

¹⁵⁰⁸ To this end, the developers can attach timestamps, expirations, and sequences to messages.

¹⁵⁰⁹ Implementers should also be aware of all the security implications resulting from the use of digital
¹⁵¹⁰ signatures in general and XML Signature in particular. When building trust into an application
¹⁵¹¹ based on a digital signature there are other technologies, such as certificate evaluation, that must
¹⁵¹² be incorporated, but these are outside the scope of this document.

¹⁵¹³ Requestors should use digital signatures to sign security tokens that do not include signatures (or
¹⁵¹⁴ other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
¹⁵¹⁵ RECOMMENDED that all relevant and immutable message content be signed by the sender.
¹⁵¹⁶ Receivers SHOULD only consider those portions of the document that are covered by the
¹⁵¹⁷ sender's signature as being subject to the security tokens in the message. Security tokens
¹⁵¹⁸ appearing in `<wsse:Security>` header elements SHOULD be signed by their issuing authority
¹⁵¹⁹ so that message receivers can have conf idence that the security tokens have not been forged or
¹⁵²⁰ altered since their issuance. It is strongly RECOMMENDED that a message sender sign any
¹⁵²¹ `<SecurityToken>` elements that it is confirming and that are not signed by their issuing
¹⁵²² authority.

¹⁵²³ Also, as described in XML Encryption, we note that the combination of signing and encryption
¹⁵²⁴ over a common data item may introduce some cryptographic vulnerability. For example,
¹⁵²⁵ encrypting digitally signed data, while leaving the digital signature in the clear, may allow plain
¹⁵²⁶ text guessing attacks. The proper usage of nonce guards aginst replay attacts.

1527   In order to *trust* Ids and timestamps, they SHOULD be signed using the mechanisms outlined in
1528   this specification. This allows readers of the IDs and timestamps information to be certain that
1529   the IDs and timestamps haven't been forged or altered in any way. It is strongly
1530   RECOMMENDED that IDs and timestamp elements be signed.

1531   Timestamps can also be used to mitigate replay attacks. Signed timestamps MAY be used to
1532   keep track of messages (possibly by caching the most recent timestamp from a specific service)
1533   and detect replays of previous messages. It is RECOMMENDED that timestamps and nonces be
1534   cached for a given period of time, as a guideline a value of five minutes can be used as a
1535   minimum to detect replays, and that timestamps older than that given period of time set be
1536   rejected. in interactive scenarios.

1537   When a password (or password equivalent) in a `<UsernameToken>` is used for authentication,
1538   the password needs to be properly protected. If the underlying transport does not provide enough
1539   protection against eavesdropping, the password SHOULD be digested as described in Section
1540   6.1.1. Even so, the password must be strong enough so that simple password guessing attacks
1541   will not reveal the secret from a captured message.

1542   In one-way message authentication, it is RECOMMENDED that the sender and the recipient re-
1543   use the elements and structure defined in this specification for proving and validating freshness of
1544   a message. It is RECOMMEND that the nonce value be unique per message (never been used
1545   as a nonce before by the sender and recipient) and use the `<wsse:Nonce>` element within the
1546   `<wsse:Security>` header. Further, the `<wsu:Timestamp>` header SHOULD be used with a
1547   `<wsu:Created>` element. It is strongly RECOMMENDED that the `<wsu:Created>`,
1548   `<wsse:Nonce>` elements be included in the signature.

1549 14 Privacy Considerations

1550 TBD

15 Acknowledgements

16 References

1558		
1559	**[DIGSIG]**	Informational RFC 2828, "Internet Security Glossary," May 2000.
1560 1561	**[Kerberos]**	J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993, http://www.ietf.org/rfc/rfc1510.txt .
1562 1563	**[KEYWORDS]**	S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997
1564 1565 1566	**[SHA-1]**	FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce / National Institute of Standards and Technology. http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt
1567	**[SOAP11]**	W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.
1568 1569	**[SOAP12]**	**W3C Working Draft, "**SOAP Version 1.2 Part 1: Messaging Framework", 26 June 2002
1570 1571	**[SOAP-SEC]**	W3C Note, "SOAP Security Extensions: Digital Signature," 06 February 2001.
1572 1573 1574	**[URI]**	T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.
1575 1576 1577	**[WS-Security]**	"Web Services Security Language", IBM, Microsoft, VeriSign, April 2002. "WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002. "WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002.
1578	**[XML-C14N]**	W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001
1579 1580	**[XML-Encrypt]**	W3C Working Draft, "XML Encryption Syntax and Processing," 04 March 2002.
1581	**[XML-ns]**	W3C Recommendation, "Namespaces in XML," 14 January 1999.
1582 1583	**[XML-Schema]**	W3C Recommendation, "XML Schema Part 1: Structures,"2 May 2001. W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.
1584 1585	**[XML Signature]**	W3C Recommendation, "XML Signature Syntax and Processing," 12 February 2002.
1586 1587 1588 1589	**[X509]**	S. Santesson, et al,"Internet X.509 Public Key Infrastructure Qualified Certificates Profile," http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent= T-REC-X.509-200003-I
1590	**[XPath]**	W3C Recommendation, "XML Path Language", 16 November 1999
1591	**[WSS-SAML]**	OASIS WorkiFeburary 2003
1592 1593	**[WSS-XrML]**	OASIS Working Draft 01, "Web Services Security XrML Token Profile, 30 January 2003

1594   **[WSS-X509]**   OASIS Working Draft 01, "Web Services Security X509 Profile, 30
1595   January 2003

1596   **[WSS-Kerberos]**   OASIS Working Draft 01, "Web Services Security Kerberos Profile, 30
1597   January 2003

1598   **[WSS-Username]**   OASIS Working Draft 01, "Web Services Security UsernameToken
1599   Profile, 23 February 2003

1600   **[XPointer]** "XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation",
1601   DeRose, Maler, Daniel, 11 September 2001.

1602

1603

Appendix A: Utility Elements and Attributes

1605 This specification defines several elements, attributes, and attribute groups which can be re-used
1606 by other specifications. This appendix provides an overview of these *utility* components. It
1607 should be noted that the detailed descriptions are provided in the specification and this appendix
1608 will reference these sections as well as calling out other aspects not documented in the
1609 specification.

1610 ## A.1. Identification Attribute

1611 There are many situations where elements within SOAP messages need to be referenced. For
1612 example, when signing a SOAP message, selected elements are included in the signature. XML
1613 Schema Part 2 provides several built-in data types that may be used for identifying and
1614 referencing elements, but their use requires that consumers of the SOAP message either to have
1615 or be able to obtain the schemas where the identity or reference mechanisms are defined. In
1616 some circumstances, for example, intermediaries, this can be problematic and not desirable.

1617 Consequently a mechanism is required for identifying and referencing elements, based on the
1618 SOAP foundation, which does not rely upon complete schema knowledge of the context in which
1619 an element is used. This functionality can be integrated into SOAP processors so that elements
1620 can be identified and referred to without dynamic schema discovery and processing.

1621 This specification specifies a namespace-qualified global attribute for identifying an element
1622 which can be applied to any element that either allows arbitrary attributes or specifically allows
1623 this attribute. This is a general purpose mechanism which can be re-used as needed.

1624 A detailed description can be found in Section 4.0 ID References.

1625 ## A.2. Timestamp Elements

1626 The specification defines XML elements which may be used to express timestamp information
1627 such as creation, expiration, and receipt. While defined in the context of messages, these
1628 elements can be re-used wherever these sorts of time statements need to be made.

1629 The elements in this specification are defined and illustrated using time references in terms of the
1630 *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this
1631 type for interoperability. It is further RECOMMENDED that all references be in UTC time for
1632 increased interoperability. If, however, other time types are used, then the *ValueType* attribute
1633 MUST be specified to indicate the data type of the time format.

1634 The following table provides an overview of these elements:

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

\<wsu:Received>	This element is used to indicate the receipt time reference associated with the enclosing context.

1635   A detailed description can be found in Section 10 Message Timestamp.

1636   A.3. General Schema Types

1637   The schema for the utility aspects of this specification also defines some general purpose
1638   schema elements. While these elements are defined in this schema for use with this
1639   specification, they are general purpose definitions that may be used by other specifications as
1640   well.

1641   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 dateTime type to include the common attributes.

1642

Appendix B: SecurityTokenReference Model

1643

1644 This appendix provides a non-normative overview of the usage and processing models for the
1645 `<wsse:SecurityTokenReference>` element.

1646 There are several motivations for introducing the `<wsse:SecurityTokenReference>`
1647 element:

1648 The XML Signature reference mechanisms are focused on "key" references rather than general
1649 token references.
1650 The XML Signature reference mechanisms utilize a fairly closed schema which limits the
1651 extensibility that can be applied.
1652 There are additional types of general reference mechanisms that are needed, but are not covered
1653 by XML Signature.
1654 There are scenarios where a reference may occur outside of an XML Signature and the XML
1655 Signature schema is not appropriate or desired.
1656 The XML Signature references may include aspects (e.g. transforms) that may not apply to all
1657 references.

1658

1659 The following use cases drive the above motivations:

1660 **Local Reference** – A security token, that is included in the message in the `<wsse:Security>`
1661 header, is associated with an XML Signature. The figure below illustrates this:
1662
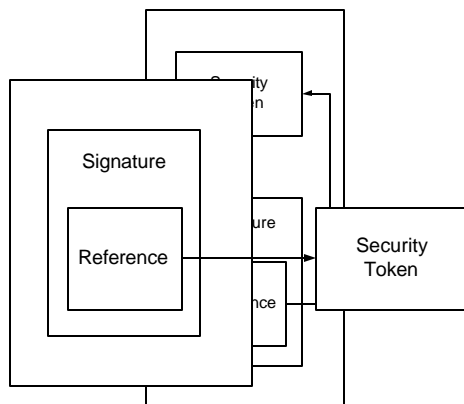
1663 **Remote Reference** – A security token, that is not included in the message but may be available
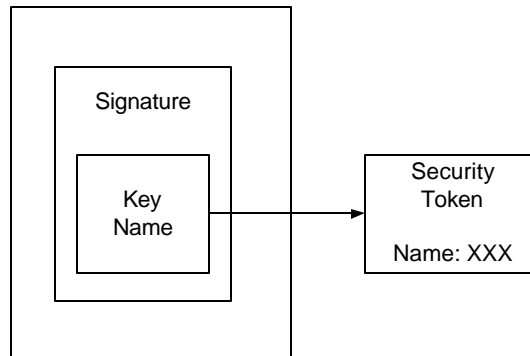1664 at a specific URI, is associated with an XML Signature. The figure below illustrates this:
1665

1666   **Key Identifier** – A security token, which is associated with an XML Signature and identified using

Signature

Key
Identifier

Security
Token

K-I(ST)
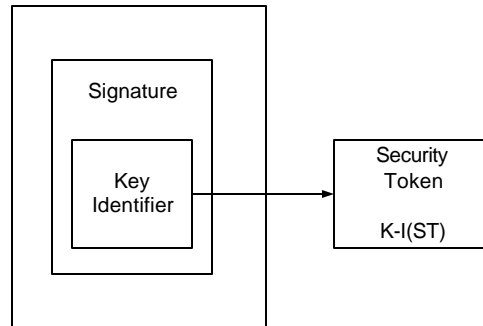
1667   a known value that is the result of a well-known function of the security token (defined by the
1668   token format or profile). The figure below illustrates this where the token is located externally:
1669
1670   **Key Name** – A security token is associated with an XML Signature and identified using a known
1671   value that represents a "name" assertion within the security token (defined by the token format or
1672   profile). The figure below illustrates this where the token is located externally:

Signature

Key
Name

Security
Token

Name: XXX

1673
1674   **Format-Specific References** – A security token is associated with an XML Signature and
1675   identified using a mechanism specific to the token (rather than the general mechanisms

MyToken
Security
Token

Signature

MyRef
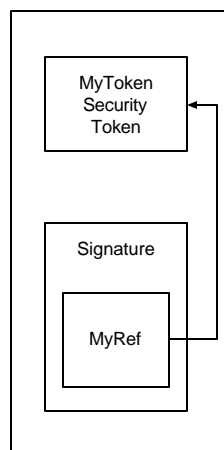
1676   described above). The figure below illustrates this:
1677
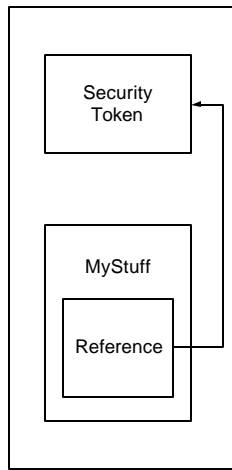
1678

1679 **Non-Signature References** – A message may contain XML that does not represent an XML
1680 signature, but may reference a security token (which may or may not be included in the
1681 message). The figure below illustrates this:

1682

1683

1684 All conformant implementations MUST be able to process the
1685 `<wsse:SecurityTokenReference>` element. However, they are not required to support all of
1686 the different types of references.

1687 The reference MAY include a *ValueType* attribute which provides a "hint" for the type of desired
1688 token.

1689 If multiple sub-elements are specified, together they describe the reference for the token.

1690 There are several challenges that implementations face when trying to interoperate:

1691 **ID References** – The underlying XML referencing mechanism using the XML base type of ID
1692 provides a simple straightforward XML element reference. However, because this is an XML
1693 type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
1694 requires the recipient to *understand* the schema. This may be an expensive task and in the
1695 general case impossible as there is no way to know the "schema location" for a specific
1696 namespace URI.
1697 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
1698 references are, by definition, unique by XML. However, other mechanisms such as "principal
1699 name" are not required to be unique and therefore such references may be unique.

1700 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
1701 information about the "key" used in the signature. For token references within signatures, it is
1702 RECOMMENDED that the `<wsse:SecurityTokenReference>` be placed within the
1703 `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
1704 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
1705 Message Security or its profiles are preferred over the mechanisms in XML Signature.

1706 The following provides additional details on the specific reference mechanisms defined in WSS:
1707 SOAP Message Security:

1708 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
1709 the security token. If only the fragment is specified, then it references the security token within
1710 the document whose *wsu:Id* matches the fragment. For non-fragment URIs, the reference is to
1711 a [potentially external] security token identified using a URI. There are no implied semantics
1712 around the processing of the URI.
1713 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
1714 by specifying a known value (identifier) for the token, which is determined by applying a special

1715 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
1716 specific security token but requires a profile or token-specific function to be specified. The
1717 *ValueType* attribute provide a *hint* as to the desired token type. The *EncodingType* attribute
1718 specifies how the unique value (identifier) is encoded. For example, a hash value may be
1719 encoded using base 64 encoding (the default).
1720 **Key Names** – The `<ds:KeyName>` element is used to reference a security token be specifying a
1721 specific value that is used to *match* identity assertion within the security token. This is a subset
1722 match and may result in multiple security tokens that match the specified name. While XML
1723 Signature doesn't imply formatting semantics, WSS: SOAP Message Security RECOMMENDS
1724 that X.509 names be specified.

1725 It is expected that, w here appropriate, profiles define if and how the reference mechanisms map
1726 to the specific token profile. Specifically, the profile should answer the following questions:

1727 What types of references can be used?
1728 How "Key Name" references map (if at all)?
1729 How "Key Identifier" references map (if at all)?
1730 Any additional profile or format-specific references?

1731

1732

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

1734

Appendix D: Notices

1735

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

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

Copyright © OASIS Open 2002. *All Rights Reserved.*

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

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

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

1765