

Web Services Security: SOAP Message Security

Working Draft 14, Monday, 30 June 2003

Document identifier:

WSS: SOAP Message Security -14

Location:

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

Editors:

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

Contributors:

TBD – Revise this list to include WSS TC contributors

Gene Thurston	AmberPoint	Chris Kaler	Microsoft Corporation
Frank Siebenlist	Argonne National Laboratory	John Shewchuk	Microsoft Corporation
Merlin Hughes	Baltimore Technologies	Prateek Mishra	Netegrity, Inc.
Irving Reid	Baltimore Technologies	Frederick Hirsch	Nokia
Pete Dapkus	BEA Systems, Inc.	Senthil Sengodan	Nokia
Hal Lockhart	BEA Systems, Inc.	Lloyd Burch	Novell
Symon Chang	CommerceOne	Ed Reed	Novell
Thomas DeMartini	ContentGuard	Charles Knouse	Obliv
Guillermo Lao	ContentGuard	Steve Anderson	OpenNetwork
TJ Pannu	ContentGuard	Vipin Samar	Oracle
Shawn Sharp	Cyclone Commerce	Jerry Schwarz	Oracle
Ganesh Vaideeswaran	Documentum	Eric Gravengaard	Reactivity
Sam Wei	Documentum	Andrew Nash	RSA Security
John Hughes	Entegrity	Rob Philpott	RSA Security

Tim Moses	Entrust	Peter Rostin	RSA Security
Toshihiro Nishimura	Fujitsu	Martijn de Boer	SAP
Tom Rutt	Fujitsu	Pete Wenzel	SeeBeyond Technology Corporation
Jason Rouault	Hewlett-Packard	Jonathan Tourzan	Sony Corporation of America
Yutaka Kudo	Hitachi	Yassir Elley	Sun Microsystems
Maryann Hondo	IBM	Jeff Hodges	Sun Microsystems
Kelvin Lawrence	IBM	Ronald Monzillo	Sun Microsystems
Anthony Nadalin	IBM	Sirish Vepa	Sybase
Don Flinn	Individual	Jan Alexander	Systinet
Phil Griffin	Individual	Michael Nguyen	The Infocomm Development Authority of Singapore
Bob Morgan	Individual	Christopher Crowhurst	Thomson Corporation
Venkat Danda	IONA	Don Adams	Tibco
Paul Cotton	Microsoft Corporation	J Weiland	US Dept of the Navy
Vijay Gajjala	Microsoft Corporation		

16 **Abstract:**

17 This specification describes enhancements to the SOAP messaging to provide quality of
 18 protection through message integrity, and single message authentication. These
 19 mechanisms can be used to accommodate a wide variety of security models and
 20 encryption technologies.

21 This specification also provides a general-purpose mechanism for associating security
 22 tokens with messages. No specific type of security token is required; it is designed to be
 23 extensible (e.g. support multiple security token formats). For example, a client might
 24 provide one format for proof of identity and provide another format for proof that they
 25 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-](mailto:wss@lists.oasis-open.org)
 35 [open.org](mailto:wss@lists.oasis-open.org) list. Others should subscribe to and send comments to the [wss-](mailto:wss-comment@lists.oasis-open.org)
 36 [comment@lists.oasis-open.org](mailto:wss-comment@lists.oasis-open.org) list. To subscribe, visit [http://lists.oasis-](http://lists.oasis-open.org/ob/adm.pl)
 37 [open.org/ob/adm.pl](http://lists.oasis-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

43	1	Introduction	5
44	1.1	Goals and Requirements	5
45	1.1.1	Requirements.....	5
46	1.1.2	Non-Goals	6
47	2	Notations and Terminology.....	7
48	2.1	Notational Conventions.....	7
49	2.2	Namespaces	7
50	2.3	Terminology	8
51	3	Message Protection Mechanisms.....	10
52	3.1	Message Security Model.....	10
53	3.2	Message Protection.....	10
54	3.3	Invalid or Missing Claims	11
55	3.4	Example	11
56	4	ID References	13
57	4.1	Id Attribute.....	13
58	4.2	Id Schema	13
59	5	Security Header	15
60	6	Security Tokens	17
61	6.1	Attaching Security Tokens	17
62	6.1.1	Processing Rules	17
63	6.1.2	Subject Confirmation	17
64	6.2	User Name Token	17
65	6.2.1	Usernames	17
66	6.3	Binary Security Tokens.....	18
67	6.3.1	Attaching Security Tokens.....	18
68	6.3.2	Encoding Binary Security Tokens	18
69	6.4	XML Tokens	20
70	6.4.1	Identifying and Referencing Security Tokens	20
71	7	Token References	21
72	7.1	SecurityTokenReference Element	21
73	7.2	Direct References	22
74	7.3	Key Identifiers.....	23
75	7.4	Embedded References	24
76	7.5	ds:KeyInfo.....	25

77	7.6 Key Names	25
78	8 Signatures	27
79	8.1 Algorithms	27
80	8.2 Signing Messages	28
81	8.3 Signing Tokens	28
82	8.4 Signature Validation	29
83	8.5 Example	30
84	9 Encryption	32
85	9.1 xenc:ReferenceList.....	32
86	9.2 xenc:EncryptedKey	33
87	9.3 Processing Rules	34
88	9.3.1 Encryption.....	34
89	9.3.2 Decryption	35
90	9.4 Decryption Transformation	35
91	10 Security Timestamps	36
92	11 Extended Example	39
93	12 Error Handling	42
94	13 Security Considerations	44
95	14 Interoperability Notes	47
96	15 Privacy Considerations.....	48
97	16 Acknowledgements.....	49
98	17 References.....	50
99	Appendix A: Utility Elements and Attributes	52
100	A.1. Identification Attribute.....	52
101	A.2. Timestamp Elements	52
102	A.3. General Schema Types	53
103	Appendix B: SecurityTokenReference Model	54
104	Appendix C: Revision History	58
105	Appendix D: Notices	59
106		

107 1 Introduction

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

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

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

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

125 1.1 Goals and Requirements

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

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

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

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

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

138 1.1.1 Requirements

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

- 141 • Multiple security token formats
- 142 • Multiple trust domains

- 143 • Multiple signature formats
- 144 • Multiple encryption technologies
- 145 • End-to-end message-level security and not just transport-level security

146 **1.1.2 Non-Goals**

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

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

152

2 Notations and Terminology

153

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

2.1 Notational Conventions

155

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

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

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

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

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

2.2 Namespaces

173

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

176 `http://schemas.xmlsoap.org/ws/2003/06/secext`
177 `http://schemas.xmlsoap.org/ws/2003/06/utility`

178 The following namespaces are used in this document:

179

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/2003/06/secext
wsu	http://schemas.xmlsoap.org/ws/2003/06/utility

180 2.3 Terminology

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

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

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

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

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

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

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

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

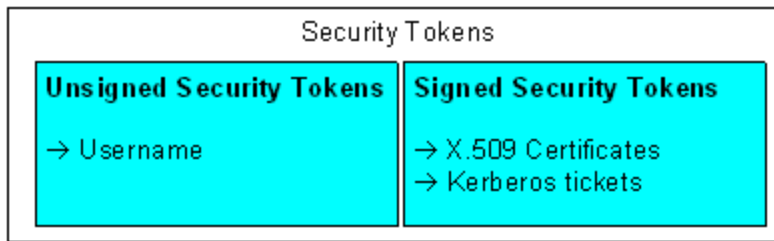
198 **Message Confidentiality** - *Message Confidentiality* is a property of the message and
199 encryption is the service or mechanism by which this property of the message is provided.

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

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

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

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



208

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

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

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

217

218

219

220 3 Message Protection Mechanisms

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

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

226 3.1 Message Security Model

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

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

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

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

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

245 3.2 Message Protection

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

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

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

257 This document defines syntax and semantics of signatures within <wsse:Security> element.
258 This document also does not specify any signature appearing outside of <wsse:Security>
259 element, if any.

260 3.3 Invalid or Missing Claims

261 The message recipient SHOULD reject a message with a signature determined to be invalid,
262 missing or unacceptable **claims** as it is an unauthorized (or malformed) message. This
263 specification provides a flexible way for the message sender to make a **claim** about the security
264 properties by associating zero or more **security tokens** with the message. An example of a
265 security **claim** is the identity of the sender; the sender can **claim** that he is Bob, known as an
266 employee of some company, and therefore he has the right to send the message.

267 3.4 Example

268 The following example illustrates the use of a username security token containing a claimed
269 security identity to establish a password derived signing key. The password is not provided in the
270 security token. The message sender combines the password with the nonce and timestamp
271 appearing in the security token to define an HMAC signing key that it then uses to sign the
272 message. The message receiver uses its knowledge of the shared secret to repeat the HMAC
273 key calculation which it uses to validate the signature and in the process confirm that the
274 message was authored by the claimed user identity. The nonce and timestamp are used in the
275 key calculation to introduce variability in the keys derived from a given password value.

```
276 (001) <?xml version="1.0" encoding="utf-8"?>
277 (002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
278       xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
279 (003)   <S:Header>
280 (004)     <wsse:Security
281           xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
282 (005)       <wsse:UsernameToken wsu:Id="MyID">
283 (006)         <wsse:Username>Zoe</wsse:Username>
284 (007)         <wsse:Nonce>FKJh...</wsse:Nonce>
285 (008)         <wsu:Created>2001-10-13T09:00:00Z</wsu:Created>
286 (009)       </wsse:UsernameToken>
287 (010)     <ds:Signature>
288 (011)       <ds:SignedInfo>
289 (012)         <ds:CanonicalizationMethod
290               Algorithm=
291                 "http://www.w3.org/2001/10/xml-exc-c14n#" />
292 (013)         <ds:SignatureMethod
293               Algorithm=
294                 "http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
295 (014)         <ds:Reference URI="#MsgBody">
296 (015)           <ds:DigestMethod
297                 Algorithm=
298                   "http://www.w3.org/2000/09/xmldsig#sha1" />
299 (016)           <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
300 (017)         </ds:Reference>
301 (018)       </ds:SignedInfo>
```

```

302      (019)          <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
303      (020)          <ds:KeyInfo>
304          (021)      <wsse:SecurityTokenReference>
305          (022)      <wsse:Reference URI="#MyID" />
306          (023)      </wsse:SecurityTokenReference>
307          (024)      </ds:KeyInfo>
308      (025)          </ds:Signature>
309      (026)          </wsse:Security>
310      (027)          </S:Header>
311      (028)          <S:Body wsu:Id="MsgBody">
312          (029)      <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
313              QQQ
314          </tru:StockSymbol>
315      (030)          </S:Body>
316      (031)          </S:Envelope>

```

317 The first two lines start the [SOAP envelope](#). Line (003) begins the headers that are associated
318 with this [SOAP message](#).

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

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

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

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

336 Line (019) specifies the signature value of the canonicalized form of the data that is being signed
337 as defined in the [XML Signature](#) specification.

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

341 Lines (028) to (030) contain the *body* (payload) of the [SOAP](#) message.

342

343

4 ID References

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

- 353 • Local ID attributes on XML Signature elements
- 354 • Local ID attributes on XML Encryption elements
- 355 • Global *wsu:id* attributes (described below) on elements

356 In addition, when signing a part of an envelope such as the body, it is RECOMMENDED that an
357 ID reference is used instead of a more general transformation, especially [XPath](#). This is to
358 simplify processing.

4.1 Id Attribute

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

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

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

4.2 Id Schema

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

378 The syntax for this attribute is as follows:

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

380 The following describes the attribute illustrated above:

381 `.../@wsu:Id`

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

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

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

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

391 `<x:myElement wsu:Id="ID1" xmlns:x="..."`
392 `xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"/>`

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

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

403

5 Security Header

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

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

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

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

428 The following illustrates the syntax of this header:

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

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

440 */wsse: Security*

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

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

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

445 */wsse:Security/{any}*

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

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

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

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

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

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

457 The optional `mustUnderstand` SOAP attribute on Security header simply means you are aware of
458 the Web Services Security: SOAP Message Security specification, and there are no implied
459 semantics.

460 6 Security Tokens

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

463 6.1 Attaching Security Tokens

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

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

469 6.1.1 Processing Rules

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

475 6.1.2 Subject Confirmation

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

479 6.2 User Name Token

480 6.2.1 Usernames

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

483 The following illustrates the syntax of this element:

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

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

488 */wsse:UsernameToken*

489 This element is used to represent a claimed identity.

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

491 A string label for this [security token](#).
492 */wsse:UsernameToken/Username*
493 This required element specifies the claimed identity.
494 */wsse:UsernameToken/Username/@{any}*
495 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
496 the `<wsse:Username>` element.
497 */wsse:UsernameToken/{any}*
498 This is an extensibility mechanism to allow different (extensible) types of security
499 information, based on a schema, to be passed.
500 */wsse:UsernameToken/@{any}*
501 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
502 added to the `UsernameToken`.
503 All compliant implementations **MUST** be able to process a `<wsse:UsernameToken>` element.
504 The following illustrates the use of this:

```
505 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
506           xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">  
507   <S:Header>  
508     ...  
509     <wsse:Security>  
510       <wsse:UsernameToken>  
511         <wsse:Username>Zoe</wsse:Username>  
512       </wsse:UsernameToken>  
513     </wsse:Security>  
514     ...  
515   </S:Header>  
516   ...  
517 </S:Envelope>
```

519 **6.3 Binary Security Tokens**

520 **6.3.1 Attaching Security Tokens**

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

524

525 **6.3.2 Encoding Binary Security Tokens**

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

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

532 The EncodingType tells how the security token is encoded, for example Base64Binary.

533 The following is an overview of the syntax:

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

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

538 /wsse:BinarySecurityToken

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

540 /wsse:BinarySecurityToken/@wsu:Id

541 An optional string label for this [security token](#).

542 /wsse:BinarySecurityToken/@ValueType

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

548 /wsse:BinarySecurityToken/@EncodingType

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

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

555 /wsse:BinarySecurityToken/@{any}

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

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

560 When a <wsse:BinarySecurityToken> is included in a signature—that is, it is referenced
561 from a <ds:Signature> element—care should be taken so that the canonicalization algorithm
562 (e.g., [Exclusive XML Canonicalization](#)) does not allow unauthorized replacement of namespace
563 prefixes of the QNames used in the attribute or element values. In particular, it is

564 RECOMMENDED that these namespace prefixes be declared within the
565 <wsse:BinarySecurityToken> element if this token does not carry the validating key (and
566 consequently it is not cryptographically bound to the [signature](#)). For example, if we wanted to
567 sign the previous example, we need to include the consumed namespace definitions.

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

```
571 <wsse:BinarySecurityToken  
572     xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext "  
573     wsu:Id="myToken"  
574     ValueType="x:MyType" xmlns:x="http://www.fabrikam123.com/x"  
575     EncodingType="wsse:Base64Binary">  
576     MIIIEZzCCA9CgAwIBAgIQEmtJZc0...  
577 </wsse:BinarySecurityToken>
```

578 6.4 XML Tokens

579 This section presents the basic principles and framework for using XML-based security tokens.
580 Subsequent specifications describe rules and processes for specific XML-based security token
581 formats.

582

583 6.4.1 Identifying and Referencing Security Tokens

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

589

590

591 7 Token References

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

593 7.1 SecurityTokenReference Element

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

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

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

604 The following illustrates the syntax of this element:

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

608 The following describes the elements defined above:

609 */wsse:SecurityTokenReference*

610 This element provides a reference to a security token.

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

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

616 */wsse:SecurityTokenReference/@wsse:Usage*

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

QName	Description
TBD	TBD

620

621 */wsse:SecurityTokenReference/{any}*

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

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

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

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

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

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

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

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

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

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

650 **Embedded References** - This allows tokens to be embedded (as opposed to a pointer to a
651 token that resides elsewhere).

652 **7.2 Direct References**

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

655 The following illustrates the syntax of this element:

```
656 <wsse:SecurityTokenReference wsu:Id="...">  
657   <wsse:Reference URI="..." ValueType="..." />  
658 </wsse:SecurityTokenReference>
```

659 The following describes the elements defined above:

660 */wsse:SecurityTokenReference/Reference*

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

662 /wsse:SecurityTokenReference/Reference/@URI

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

665 /wsse:SecurityTokenReference/Reference/@ValueType

666 This optional attribute specifies a QName that is used to identify the *type* of token being
667 referenced (see <wsse:BinarySecurityToken>). This specification does not define
668 any processing rules around the usage of this attribute, however, specifications for
669 individual token types MAY define specific processing rules and semantics around the
670 value of the URI and how it SHALL be interpreted. If this attribute is not present, the URI
671 SHALL be processed as a normal URI. The usage of ValueType is RECOMMENDED for
672 local URIs.

673 /wsse:SecurityTokenReference/Reference/{any}

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

676 /wsse:SecurityTokenReference/Reference/@{any}

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

679 The following illustrates the use of this element:

```
680 <wsse:SecurityTokenReference
681     xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
682     <wsse:Reference
683         URI="http://www.fabrikaml23.com/tokens/Zoe" />
684 </wsse:SecurityTokenReference>
```

685 7.3 Key Identifiers

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

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

695 The processing model assumes that the key identifier for a security token is constant.
696 Consequently, processing a key identifier is simply looking for a security token whose key
697 identifier matches a given specified constant.

698 The following is an overview of the syntax:

```
699 <wsse:SecurityTokenReference>
700     <wsse:KeyIdentifier wsu:Id="..."
701         ValueType="..."
702         EncodingType="...">
```

```

703     ...
704     </wsse:KeyIdentifier>
705 </wsse:SecurityTokenReference>

```

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

707 */wsse:SecurityTokenReference/KeyIdentifier*

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

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

710 An optional string label for this identifier.

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

712 The optional ValueType attribute is used to indicate the type of KeyIdentifier being used.
 713 Each token profile specifies the KeyIdentifier types that may be used to refer to tokens of
 714 that type. It also specifies the critical semantics of the identifier, such as whether the
 715 KeyIdentifier is unique to the key or the token. Any value specified for binary security
 716 tokens, or any XML token element QName can be specified here.

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

718 The optional EncodingType attribute is used to indicate, using a QName, the encoding
 719 format of the KeyIdentifier (e.g., wsse:Base64Binary). The base values defined in this
 720 specification are used:

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

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

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

724 7.4 Embedded References

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

728 The following is an overview of the syntax:

```

729 <wsse:SecurityTokenReference>
730   <wsse:Embedded wsu:Id="...">
731     ...
732   </wsse:Embedded>
733 </wsse:SecurityTokenReference>

```

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

735 */wsse:SecurityTokenReference/Embedded*

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

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

739 An optional string label for this element.

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

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

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

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

746 The following example illustrates embedding a SAML assertion:

```
747 <S:Envelope>  
748   <S:Header>  
749     <wsse:Security>  
750       ...  
751       <wsse:SecurityTokenReference>  
752         <wsse:Embedded wsu:Id="tok1">  
753           <saml:Assertion xmlns:saml="...">  
754             ...  
755           </saml:Assertion xmlns:saml="...">  
756         </wsse:Embedded>  
757       </wsse:SecurityTokenReference>  
758     </wsse:Security>  
759   </S:Header>  
760   ...  
761 </S:Body>
```

763 7.5 ds:KeyInfo

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

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

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

773 7.6 Key Names

774 It is strongly RECOMMENDED to use key identifiers. However, if key names are used, then it is
775 strongly RECOMMENDED that `<ds:KeyName>` elements conform to the attribute names in

776 section 2.3 of RFC 2253 (this is recommended by XML Signature for <x509SubjectName>) for
777 interoperability.

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

779 EmailAddress=ckaler@microsoft.com

780

781

8 Signatures

782
783
784

Message senders may want to enable message recipients to determine whether a message was altered in transit and to verify that the claims in a particular [security token](#) apply to the sender of the message.

785
786
787
788
789
790

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

791
792
793
794

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

795
796
797
798
799
800
801
802
803
804
805
806

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

807

All compliant implementations MUST be able to support the [XML Signature](#) standard.

808

8.1 Algorithms

809
810

This specification builds on [XML Signature](#) and therefore has the same algorithm requirements as those specified in the [XML Signature](#) specification.

811
812

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

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

813 The [Exclusive XML Canonicalization](#) algorithm addresses the pitfalls of general canonicalization
814 that can occur from *leaky* namespaces with pre-existing signatures.
815 Finally, if a sender wishes to sign a message before encryption, they should alter the order of the
816 signature and encryption elements inside of the `<wsse:Security>` header.

817 **8.2 Signing Messages**

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

824 [SOAP](#) applications MUST satisfy the following conditions:

825 The application MUST be capable of processing the required elements defined in the [XML
826 Signature](#) specification.

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

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

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

841 Due to security concerns with namespaces, this specification strongly RECOMMENDS the use of
842 the "[Exclusive XML Canonicalization](#)" algorithm or another canonicalization algorithm that
843 provides equivalent or greater protection.

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

846 **8.3 Signing Tokens**

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

851 This specification allows different tokens to have their own unique reference mechanisms which
852 are specified in their profile as extensions to the <SecurityTokenReference> element. This
853 element provides a uniform referencing mechanism that is guaranteed to work with all token
854 formats. Consequently, this specification defines a new reference option for XML Signature: the
855 STR Dereference Transform.

856 This transform is specified by the URI <http://schemas.xmlsoap.org/2003/06/STR-Transform> and
857 when applied to a <SecurityTokenReference> element it means that the output is the token
858 referenced by the <SecurityTokenReference> element not the element itself.

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

865 The following illustrates an example of this transformation which references a token contained
866 w within the message envelope:

```
867 ...  
868 <wsse:SecurityTokenReference wsu:Id="Str1">  
869 ...  
870 </wsse:SecurityTokenReference>  
871 ...  
872 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">  
873   <SignedInfo>  
874     ...  
875     <Reference URI="#Str1">  
876       <Transforms>  
877         <ds:Transform  
878           Algorithm=  
879             "http://schemas.xmlsoap.org/2003/06/STR-Transform" />  
880       </Transforms>  
881       <DigestMethod Algorithm=  
882         "http://www.w3.org/2000/09/xmldsig#sha1" />  
883       <DigestValue>...</DigestValue>  
884     </Reference>  
885   </SignedInfo>  
886   <SignatureValue></SignatureValue>  
887 </Signature>  
888 ...
```

889 **8.4 Signature Validation**

890 The validation of a <ds:Signature> element inside an <wsse:Security> header block
891 SHALL fail if

- 892 • the syntax of the content of the element does not conform to this specification, or
- 893 • the validation of the [signature](#) contained in the element fails according to the core
894 validation of the [XML Signature](#) specification, or

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

898 If the validation of the signature element fails, applications MAY report the failure to the sender
899 using the fault codes defined in [Section 12 Error Handling](#).

900 8.5 Example

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

```
903 <?xml version="1.0" encoding="utf-8"?>
904 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
905           xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
906           xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
907           xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
908   <S:Header>
909     <wsse:Security>
910       <wsse:BinarySecurityToken
911         ValueType="wsse:X509v3"
912         EncodingType="wsse:Base64Binary"
913         wsu:Id="X509Token">
914         MIEZzCCA9CgAwIBAgIQEmtJZc0rqKh5i...
915       </wsse:BinarySecurityToken>
916       <ds:Signature>
917         <ds:SignedInfo>
918           <ds:CanonicalizationMethod Algorithm=
919             "http://www.w3.org/2001/10/xml-exc-c14n#" />
920           <ds:SignatureMethod Algorithm=
921             "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
922           <ds:Reference URI="#myBody">
923             <ds:Transforms>
924               <ds:Transform Algorithm=
925                 "http://www.w3.org/2001/10/xml-exc-c14n#" />
926             </ds:Transforms>
927             <ds:DigestMethod Algorithm=
928               "http://www.w3.org/2000/09/xmldsig#sha1" />
929             <ds:DigestValue>EULddytSol...</ds:DigestValue>
930           </ds:Reference>
931         </ds:SignedInfo>
932         <ds:SignatureValue>
933         BL8jdfToEb1l/vXcMZNNjPOV...
934         </ds:SignatureValue>
935         <ds:KeyInfo>
936           <wsse:SecurityTokenReference>
937             <wsse:Reference URI="#X509Token" />
938           </wsse:SecurityTokenReference>
939         </ds:KeyInfo>
940       </ds:Signature>
941     </wsse:Security>
942   </S:Header>
943   <S:Body wsu:Id="myBody">
```

```
944     <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
945         QQQ
946     </tru:StockSymbol>
947 </S:Body>
948 </S:Envelope>
```

949

9 Encryption

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

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

964 All compliant implementations MUST be able to support the [XML Encryption](#) standard.

9.1 xenc:ReferenceList

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

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

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

```
981 <S:Envelope  
982   xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
983   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
984   xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext "  
985   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">  
986   <S:Header>  
987     <wsse:Security>
```

```

988         <xenc:ReferenceList>
989             <xenc:DataReference URI="#bodyID" />
990         </xenc:ReferenceList>
991     </wsse:Security>
992 </S:Header>
993 <S:Body>
994     <xenc:EncryptedData Id="bodyID">
995         <ds:KeyInfo>
996             <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
997         </ds:KeyInfo>
998         <xenc:CipherData>
999             <xenc:CipherValue>...</xenc:CipherValue>
1000         </xenc:CipherData>
1001     </xenc:EncryptedData>
1002 </S:Body>
1003 </S:Envelope>

```

1004 9.2 xenc:EncryptedKey

1005 When the encryption step involves encrypting elements or element contents within a [SOAP](#)
1006 envelope with a symmetric key, which is in turn to be encrypted by the recipient's key and
1007 embedded in the message, `<xenc:EncryptedKey>` MAY be used for carrying such an
1008 encrypted key. This sub-element SHOULD have a manifest, that is, an
1009 `<xenc:ReferenceList>` element, in order for the recipient to know the portions to be
1010 decrypted with this key. An element or element content to be encrypted by this encryption step
1011 MUST be replaced by a corresponding `<xenc:EncryptedData>` according to [XML Encryption](#).
1012 All the `<xenc:EncryptedData>` elements created by this encryption step SHOULD be listed in
1013 the `<xenc:ReferenceList>` element inside this sub-element.

1014 This construct is useful when encryption is done by a randomly generated symmetric key that is
1015 in turn encrypted by the recipient's public key. The following illustrates the use of this element:

```

1016 <S:Envelope
1017     xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1018     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1019     xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
1020     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1021     <S:Header>
1022         <wsse:Security>
1023             <xenc:EncryptedKey>
1024                 <xenc:EncryptionMethod Algorithm="..." />
1025                 <ds:KeyInfo>
1026                     <wsse:SecurityTokenReference>
1027                         <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1028                             ValueType="wsse:X509v3">MIGfMa0GCSq...
1029                     </wsse:KeyIdentifier>
1030                 </wsse:SecurityTokenReference>
1031                 </ds:KeyInfo>
1032                 <xenc:CipherData>
1033                     <xenc:CipherValue>...</xenc:CipherValue>
1034                 </xenc:CipherData>
1035             </xenc:EncryptedKey>

```

```

1036         <xenc:DataReference URI="#bodyID" />
1037         </xenc:ReferenceList>
1038         </xenc:EncryptedKey>
1039     </wsse:Security>
1040 </S:Header>
1041 <S:Body>
1042     <xenc:EncryptedData Id="bodyID">
1043         <xenc:CipherData>
1044             <xenc:CipherValue>...</xenc:CipherValue>
1045         </xenc:CipherData>
1046     </xenc:EncryptedData>
1047 </S:Body>
1048 </S:Envelope>

```

1049 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in
1050 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that
1051 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

1052 9.3 Processing Rules

1053 Encrypted parts or attachments to the SOAP message using one of the sub-elements defined
1054 above MUST be in compliance with the XML Encryption specification. An encrypted SOAP
1055 envelope MUST still be a valid SOAP envelope. The message creator MUST NOT encrypt the
1056 `<S:Envelope>`, `<S:Header>`, or `<S:Body>` elements but MAY encrypt child elements of
1057 either the `<S:Header>` and `<S:Body>` elements. Multiple steps of encryption MAY be added
1058 into a single `<Security>` header block if they are targeted for the same recipient.

1059 When an element or element content inside a SOAP envelope (e.g. of the contents of `<S:Body>`)
1060 is to be encrypted, it MUST be replaced by an `<xenc:EncryptedData>`, according to XML
1061 Encryption and it SHOULD be referenced from the `<xenc:ReferenceList>` element created
1062 by this encryption step. This specification allows placing the encrypted octet stream in an
1063 attachment. For example, if an `<xenc:EncryptedData>` element in an `<S:Body>` element has
1064 `<xenc:CipherReference>` that refers to an attachment, then the decrypted octet stream
1065 SHALL replace the `<xenc:EncryptedData>`. However, if the `<xenc:EncryptedData>`
1066 element is located in the `<Security>` header block and it refers to an attachment, then the
1067 decrypted octet stream MUST replace the encrypted octet stream in the attachment.

1068 9.3.1 Encryption

1069 The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1070 this specification are listed below (note that use of `<xenc:ReferenceList>` is
1071 RECOMMENDED).

1072 Create a new SOAP envelope.

1073 Create a `<Security>` header

1074 Create an `<xenc:ReferenceList>` sub-element, an `<xenc:EncryptedKey>` sub-element, or
1075 an `<xenc:EncryptedData>` sub-element in the `<Security>` header block (note that if the
1076 SOAP "role" and "mustUnderstand" attributes are different, then a new header block may be
1077 necessary), depending on the type of encryption.

1078 Locate data items to be encrypted, i.e., XML elements, element contents within the target SOAP
1079 envelope, and attachments.

1080 Encrypt the data items as follows: For each XML element or element content within the target
1081 SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification.
1082 Each selected original element or element content MUST be removed and replaced by the
1083 resulting <xenc:EncryptedData> element. For an attachment, the contents MUST be replaced
1084 by encrypted cipher data as described in section 9.3 Signature Validation.

1085 The optional <ds:KeyInfo> element in the <xenc:EncryptedData> element MAY reference
1086 another <ds:KeyInfo> element. Note that if the encryption is based on an attached security
1087 token, then a <SecurityTokenReference> element SHOULD be added to the
1088 <ds:KeyInfo> element to facilitate locating it.

1089 Create an <xenc:DataReference> element referencing the generated
1090 <xenc:EncryptedData> elements. Add the created <xenc:DataReference> element to the
1091 <xenc:ReferenceList>.

1092 9.3.2 Decryption

1093 On receiving a SOAP envelope containing encryption header elements, for each encryption
1094 header element the following general steps should be processed (non-normative):

1095 Locate the <xenc:EncryptedData> items to be decrypted (possibly using the
1096 <xenc:ReferenceList>).

1097 Decrypt them as follows: For each element in the target SOAP envelope, decrypt it according to
1098 the processing rules of the XML Encryption specification and the processing rules listed above.

1099 If the decrypted data is part of an attachment and MIME types were used, then revise the MIME
1100 type of the attachment to the original MIME type (if one exists).

1101 If the decryption fails for some reason, applications MAY report the failure to the sender using the
1102 fault code defined in Section 12 Error Handling.

1103 9.4 Decryption Transformation

1104 The ordering semantics of the <wsse:Security> header are sufficient to determine if
1105 signatures are over encrypted or unencrypted data. However, when a signature is included in
1106 one <wsse:Security> header and the encryption data is in another <wsse:Security>
1107 header, the proper processing order may not be apparent.

1108 If the sender wishes to sign a message that MAY subsequently be encrypted by an intermediary
1109 then the sender MAY use the Decryption Transform for XML Signature to explicitly specify the
1110 order of decryption.

1111

10 Security Timestamps

1112

1113 It is often important for the recipient to be able to determine the *freshness* of security semantics.
1114 In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.

1115 This specification does not provide a mechanism for synchronizing time. The assumption is that
1116 time is trusted or additional mechanisms, not described here, are employed to prevent replay.

1117 This specification defines and illustrates time references in terms of the *dateTime* type defined in
1118 XML Schema. It is RECOMMENDED that all time references use this type. It is further
1119 RECOMMENDED that all references be in UTC time. If, however, other time types are used,
1120 then the *ValueType* attribute (described below) MUST be specified to indicate the data type of the
1121 time format. Requestors and receivers SHOULD NOT rely on other applications supporting time
1122 resolution finer than milliseconds. Implementations MUST NOT generate time instants that
1123 specify leap seconds.

1124 The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and
1125 expiration times of the security semantics in a message.

1126 All times SHOULD be in UTC format as specified by the [XML Schema](#) type (`dateTime`). It should
1127 be noted that times support time precision as defined in the [XML Schema](#) specification.

1128 The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and
1129 may only be present at most once per header (that is, per [SOAP](#) role).

1130 The ordering within the element is as illustrated below. The ordering of elements in this header is
1131 fixed and MUST be preserved by intermediaries.

1132 To preserve overall integrity of each `<wsu:Timestamp>` element, it is strongly RECOMMENDED
1133 that each [SOAP](#) role create or update the appropriate `<wsu:Timestamp>` element destined to
1134 itself (that is, a `<wsse:Security>` header whose actor/role is itself).

1135 The schema outline for the `<wsu:Timestamp>` element is as follows:

```
1136 <wsu:Timestamp wsu:Id="...">  
1137   <wsu:Created ValueType="...">...</wsu:Created>  
1138   <wsu:Expires ValueType="...">...</wsu:Expires>  
1139   ...  
1140 </wsu:Timestamp>
```

1141 The following describes the attributes and elements listed in the schema above:

1142 */wsu:Timestamp*

1143 This is the header for indicating message timestamps.

1144 */wsu:Timestamp/Created*

1145 This represents the [creation time](#) of the security semantics. This element is optional, but
1146 can only be specified once in a `Timestamp` element. Within the `SOAP` processing
1147 model, creation is the instant that the infoSet is serialized for transmission. The creation
1148 time of the message SHOULD NOT differ substantially from its transmission time. The
1149 difference in time should be minimized.

1150 */wsu:Timestamp/wsu:Created/@ValueType*
 1151 This optional attribute specifies the type of the time data. This is specified as the XML
 1152 Schema type. The default value is `xsd:dateTime`.
 1153 */wsu:Timestamp/Expires*
 1154 This represents the [expiration](#) of the security semantics. This is optional, but can appear
 1155 at most once in a `Timestamp` element. Upon expiration, the requestor asserts that its
 1156 security semantics are no longer valid. It is strongly RECOMMENDED that recipients
 1157 (anyone who processes this message) discard (ignore) any message whose security
 1158 semantics have passed their expiration. A Fault code (`wsu:MessageExpired`) is provided
 1159 if the recipient wants to inform the requestor that its security semantics were expired. A
 1160 service MAY issue a Fault indicating the security semantics have expired.
 1161 */wsu:Timestamp/wsu:Expires/@ValueType*
 1162 This optional attribute specifies the type of the time data. This is specified as the XML
 1163 Schema type. The default value is `xsd:dateTime`.
 1164 */wsu:Timestamp/{any}*
 1165 This is an extensibility mechanism to allow additional elements to be added to the
 1166 element.
 1167 */wsu:Timestamp/@wsu:Id*
 1168 This optional attribute specifies an XML Schema ID that can be used to reference this
 1169 element (the timestamp). This is used, for example, to reference the timestamp in a XML
 1170 Signature.
 1171 */wsu:Timestamp/@{any}*
 1172 This is an extensibility mechanism to allow additional attributes to be added to the
 1173 element.
 1174 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
 1175 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
 1176 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
 1177 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
 1178 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
 1179 judgment of the requestor's likely current clock time by means not described in this specification,
 1180 for example an out-of-band clock synchronization protocol. The recipient may also use the
 1181 creation time and the delays introduced by intermediate [SOAP](#) roles to estimate the degree of
 1182 clock skew.

1183 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```

1184 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1185           xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
1186           xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
1187   <S:Header>
1188     <wsse:Security>
1189       <wsu:Timestamp wsu:Id="timestamp">
1190         <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
1191         <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
1192       </wsu:Timestamp>

```

1193
1194
1195
1196
1197
1198
1199
1200

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

11 Extended Example

1201

1202 The following sample message illustrates the use of security tokens, signatures, and encryption.
1203 For this example, the timestamp and the message body are signed prior to encryption. The
1204 decryption transformation is not needed as the signing/encryption order is specified within the
1205 <wsse:Security> header.

```
1206 (001) <?xml version="1.0" encoding="utf-8"?>
1207 (002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1208         xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1209         xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
1210         xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
1211         xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1212 (003)   <S:Header>
1213 (004)     <wsse:Security>
1214 (005)       <wsu:Timestamp>
1215 (006)         <wsu:Created
1216 (007)           wsu:Id="T0">2001-09-13T08:42:00Z</wsu:Created>
1217 (008)         </wsu:Timestamp>
1218 (009)
1219 (010)       <wsse:BinarySecurityToken
1220             ValueType="wsse:X509v3"
1221             wsu:Id="X509Token"
1222             EncodingType="wsse:Base64Binary">
1223 (011) MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1224 (012) </wsse:BinarySecurityToken>
1225 (013) <xenc:EncryptedKey>
1226 (014)   <xenc:EncryptionMethod Algorithm=
1227         "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
1228 (015)   <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1229         ValueType="wsse:X509v3">MIGfMa0GCSq...
1230 (017) </wsse:KeyIdentifier>
1231 (018)   <xenc:CipherData>
1232 (019)     <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1233 (020)     </xenc:CipherValue>
1234 (021)   </xenc:CipherData>
1235 (022)   <xenc:ReferenceList>
1236 (023)     <xenc:DataReference URI="#enc1"/>
1237 (024)   </xenc:ReferenceList>
1238 (025) </xenc:EncryptedKey>
1239 (026) <ds:Signature>
1240 (027)   <ds:SignedInfo>
1241 (028)     <ds:CanonicalizationMethod
1242         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1243 (029)     <ds:SignatureMethod
1244         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
1245 (039)     <ds:Reference URI="#T0">
1246 (031)       <ds:Transforms>
1247 (032)         <ds:Transform
```

```

1248 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1249 (033) </ds:Transforms>
1250 (034) <ds:DigestMethod
1251 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1252 (035) <ds:DigestValue>LyLsF094hPi4wPU...
1253 (036) </ds:DigestValue>
1254 (037) </ds:Reference>
1255 (038) <ds:Reference URI="#body">
1256 (039) <ds:Transforms>
1257 (040) <ds:Transform
1258 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1259 (041) </ds:Transforms>
1260 (042) <ds:DigestMethod
1261 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1262 (043) <ds:DigestValue>LyLsF094hPi4wPU...
1263 (044) </ds:DigestValue>
1264 (045) </ds:Reference>
1265 (046) </ds:SignedInfo>
1266 (047) <ds:SignatureValue>
1267 (048) Hp1ZkmFZ/2kQLXDJbchm5gK...
1268 (049) </ds:SignatureValue>
1269 (050) <ds:KeyInfo>
1270 (051) <wsse:SecurityTokenReference>
1271 (052) <wsse:Reference URI="#X509Token" />
1272 (053) </wsse:SecurityTokenReference>
1273 (054) </ds:KeyInfo>
1274 (055) </ds:Signature>
1275 (056) </wsse:Security>
1276 (057) </S:Header>
1277 (058) <S:Body wsu:Id="body">
1278 (059) <xenc:EncryptedData
1279 Type="http://www.w3.org/2001/04/xmlenc#Element"
1280 wsu:Id="enc1">
1281 (060) <xenc:EncryptionMethod
1282 Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-
1283 cbc" />
1284 (061) <xenc:CipherData>
1285 (062) <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1286 (063) </xenc:CipherValue>
1287 (064) </xenc:CipherData>
1288 (065) </xenc:EncryptedData>
1289 (066) </S:Body>
1290 (067) </S:Envelope>

```

1291 Let's review some of the key sections of this example:

1292 Lines (003)-(057) contain the SOAP message headers.

1293 Lines (004)-(056) represent the <wsse:Security> header block. This contains the security-
1294 related information for the message.

1295 Lines (005)-(008) specify the timestamp information. In this case it indicates the creation time of
1296 the security semantics.

1297 Lines (010)-(012) specify a [security token](#) that is associated with the message. In this case, it
1298 specifies an [X.509](#) certificate that is encoded as Base64. Line (011) specifies the actual Base64
1299 encoding of the certificate.

1300 Lines (013)-(025) specify the key that is used to encrypt the body of the message. Since this is a
1301 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
1302 encrypt the key. Lines (015)-(017) specify the name of the key that was used to encrypt the
1303 symmetric key. Lines (018)-(021) specify the actual encrypted form of the symmetric key. Lines
1304 (022)-(024) identify the encryption block in the message that uses this symmetric key. In this
1305 case it is only used to encrypt the body (Id="enc1").

1306 Lines (026)-(055) specify the digital signature. In this example, the signature is based on the
1307 [X.509](#) certificate. Lines (027)-(046) indicate what is being signed. Specifically, Line (039)
1308 references the creation timestamp and line (038) references the message body.

1309 Lines (047)-(049) indicate the actual signature value – specified in Line (042).

1310 Lines (051)-(053) indicate the key that was used for the signature. In this case, it is the [X.509](#)
1311 certificate included in the message. Line (052) provides a URI link to the Lines (010)-(012).

1312 The body of the message is represented by Lines (056)-(066).

1313 Lines (059)-(065) represent the encrypted metadata and form of the body using [XML Encryption](#).
1314 Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line
1315 (060) specifies the encryption algorithm – Triple-DES in this case. Lines (062)-(063) contain the
1316 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
1317 key as the key references this encryption – Line (023).

12 Error Handling

1318

1319 There are many circumstances where an *error* can occur while processing security information.
1320 For example:

- 1321 • Invalid or unsupported type of security token, signing, or encryption
- 1322 • Invalid or unauthenticated or unauthenticatable security token
- 1323 • Invalid signature
- 1324 • Decryption failure
- 1325 • Referenced security token is unavailable
- 1326 • Unsupported namespace

1327 If a service does not perform its normal operation because of the contents of the Security header,
1328 then that MAY be reported using SOAP's Fault Mechanism. This specification does not mandate
1329 that faults be returned as this could be used as part of a denial of service or cryptographic
1330 attack. We combine signature and encryption failures to mitigate certain types of attacks.

1331 If a failure is returned to a sender then the failure MUST be reported using the [SOAP Fault](#)
1332 mechanism. The following tables outline the predefined security fault codes. The "unsupported"
1333 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

1334 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

1335

1336 It is strongly RECOMMENDED that messages include digitally signed elements to allow message
1337 recipients to detect replays of the message when the messages are exchanged via an open
1338 network. These can be part of the message or of the headers defined from other [SOAP](#)
1339 extensions. Four typical approaches are:

- 1340 • Timestamp
- 1341 • Sequence Number
- 1342 • Expirations
- 1343 • Message Correlation

1344 This specification defines the use of [XML Signature](#) and [XML Encryption](#) in [SOAP](#) headers. As
1345 one of the building blocks for securing [SOAP](#) messages, it is intended to be used in conjunction
1346 with other security techniques. Digital signatures need to be understood in the context of other
1347 security mechanisms and possible threats to an entity.

1348 Digital signatures alone do not provide message authentication. One can record a signed
1349 message and resend it (a replay attack). To prevent this type of attack, digital signatures must be
1350 combined with an appropriate means to ensure the uniqueness of the message, such as
1351 timestamps or sequence numbers (see earlier section for additional details). The proper usage of
1352 nonce guards against replay attacks.

1353 When digital signatures are used for verifying the claims pertaining to the sending entity, the
1354 sender must demonstrate knowledge of the confirmation key. One way to achieve this is to use a
1355 challenge-response type of protocol. Such a protocol is outside the scope of this document.

1356 To this end, the developers can attach timestamps, expirations, and sequences to messages.

1357 Implementers should also be aware of all the security implications resulting from the use of digital
1358 signatures in general and [XML Signature](#) in particular. When building trust into an application
1359 based on a digital signature there are other technologies, such as certificate evaluation, that must
1360 be incorporated, but these are outside the scope of this document.

1361 Implementers should be aware of the possibility of a token substitution attack. In any situation
1362 where a digital signature is verified by reference to a token provided in the message, which
1363 specifies the key, it may be possible for an unscrupulous sender to later claim that a different
1364 token, containing the same key, but different information was intended.

1365 An example of this would be a user who had multiple X.509 certificates issued relating to the
1366 same key pair but with different attributes, constraints or reliance limits. Note that the signature of
1367 the token by its issuing authority does not prevent this attack. Nor can an authority effectively
1368 prevent a different authority from issuing a token over the same key if the user can prove
1369 possession of the secret.

1370 The most straightforward counter to this attack is to insist that the token (or its unique identifying
1371 data) be included under the signature of the sender. If the nature of the application is such that
1372 the contents of the token are irrelevant, assuming it has been issued by a trusted authority, this

1373 attack may be ignored. However because application semantics may change over time, best
1374 practice is to prevent this attack.

1375 Requestors should use digital signatures to sign security tokens that do not include signatures (or
1376 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly
1377 RECOMMENDED that all relevant and immutable message content be signed by the sender.
1378 Receivers SHOULD only consider those portions of the document that are covered by the
1379 sender's signature as being subject to the security tokens in the message. Security tokens
1380 appearing in `<wsse:Security>` header elements SHOULD be signed by their issuing authority
1381 so that message receivers can have confidence that the security tokens have not been forged or
1382 altered since their issuance. It is strongly RECOMMENDED that a message sender sign any
1383 `<SecurityToken>` elements that it is confirming and that are not signed by their issuing
1384 authority.

1385 When a requester provides, within the request, a Public Key to be used to encrypt the response,
1386 it is possible that an attacker in the middle may substitute a different Public Key, thus allowing the
1387 attacker to read the response. The best way to prevent this attack is to bind the encryption key in
1388 some way to the request. One simple way of doing this is to use the same key pair to sign the
1389 request as to encrypt the response. However, if policy requires the use of distinct key pairs for
1390 signing and encryption, then the Public Key provided in the request should be included under the
1391 signature of the request.

1392 Also, as described in [XML Encryption](#), we note that the combination of signing and encryption
1393 over a common data item may introduce some cryptographic vulnerability. For example,
1394 encrypting digitally signed data, while leaving the digital signature in the clear, may allow plain
1395 text guessing attacks. The proper usage of nonce guards against replay attacks.

1396 In order to *trust* IDs and timestamps, they SHOULD be signed using the mechanisms outlined in
1397 this specification. This allows readers of the IDs and timestamps information to be certain that
1398 the IDs and timestamps haven't been forged or altered in any way. It is strongly
1399 RECOMMENDED that IDs and timestamp elements be signed.

1400 Timestamps can also be used to mitigate replay attacks. Signed timestamps MAY be used to
1401 keep track of messages (possibly by caching the most recent timestamp from a specific service)
1402 and detect replays of previous messages. It is RECOMMENDED that timestamps and nonce be
1403 cached for a given period of time, as a guideline a value of five minutes can be used as a
1404 minimum to detect replays, and that timestamps older than that given period of time set be
1405 rejected in interactive scenarios.

1406 When a password (or password equivalent) in a `<UsernameToken>` is used for authentication,
1407 the password needs to be properly protected. If the underlying transport does not provide enough
1408 protection against eavesdropping, the password SHOULD be digested as described in the Web
1409 Services Security: Username Token Profile Document. Even so, the password must be strong
1410 enough so that simple password guessing attacks will not reveal the secret from a captured
1411 message.

1412 When a password is encrypted in addition to the normal threats against any encryption, two
1413 password-specific threats must be considered: replay and guessing. If an attacker can
1414 impersonate a user by replaying an encrypted or hashed password, then learning the actual
1415 password is not necessary. One method of preventing replay is to use a nonce as mentioned
1416 previously. Generally it is also necessary to use a timestamp to put a ceiling on the number of

1417 previous nonces that must be stored. However, in order to be effective the nonce and timestamp
1418 must be signed. If the signature is also over the password itself, prior to encryption, then it would
1419 be a simple matter to use the signature to perform an offline guessing attack against the
1420 password. This threat can be countered in any of several ways including: don't include the
1421 password under the signature (the password will be verified later) or sign the encrypted
1422 password.

1423 In one-way message authentication, it is RECOMMENDED that the sender and the recipient re-
1424 use the elements and structure defined in this specification for proving and validating freshness of
1425 a message. It is RECOMMEND that the nonce value be unique per message (never been used
1426 as a nonce before by the sender and recipient) and use the <wsse:Nonce> element within the
1427 <wsse:Security> header. Further, the <wsu:Timestamp> header SHOULD be used with a
1428 <wsu:Created> element. It is strongly RECOMMENDED that the <wsu:Created> ,
1429 <wsse:Nonce> elements be included in the signature.

14 Interoperability Notes

1430

1431 Based on interoperability experiences with this and similar specifications, the following list
1432 highlights several common areas where interoperability issues have been discovered. Care
1433 should be taken when implementing to avoid these issues. It should be noted that some of these
1434 may seem "obvious", but have been problematic during testing.

- 1435 • Key Identifiers: Make sure you understand the algorithm and how it is applied to security
1436 tokens.
- 1437 • EncryptedKey: The EncryptedKey element from XML Encryption requires a Type attribute
1438 whose value is one of a pre-defined list of values. Ensure that a correct value is used.
- 1439 • Encryption Padding: Both RSA random padding and the XML Encryption random block
1440 cipher padding have both cause issues, be careful to follow the specifications exactly.
- 1441 • IDs: The specification recognizes on three specific ID elements: the global wsu:Id
1442 attribute and the local Id attributes on XML Signature and XML Encryption elements
1443 (because the latter two do not allow global attributes). If any other element does not
1444 allow global attributes, it cannot be directly signed using an ID reference. Note that the
1445 global attribute wsu:Id MUST carry the namespace specification.
- 1446 • Time Formats: This specification uses a restricted version of the XML Schema dateTime
1447 element. Take care to ensure compliance with the specified restrictions.
- 1448 • Byte Order Marker (BOM): Some implementations have problems processing the BOM
1449 marker. It is suggested that usage of this be optional.
- 1450 • SOAP, WSDL, HTTP: Various interoperability issues have been seen with incorrect
1451 SOAP, WSDL, and HTTP semantics being applied. Care should be taken to carefully
1452 adhere to these specifications and any interoperability guidelines that are available.

1453 **15 Privacy Considerations**

1454 If messages contain data that is sensitive or personal in nature or for any reason should not be
1455 visible to parties other than the sender and authorized recipients, the use of encryption, as
1456 described in this specification, is strongly RECOMMENDED.

1457 This specification DOES NOT define mechanisms for making privacy statements or requirements.

1458 **16 Acknowledgements**

1459 This specification was developed as a result of joint work of many individuals from the WSS TC
1460 including: TBD

1461 The input specifications for this document were developed as a result of joint work with many
1462 individuals and teams, including: Keith Ballinger, Microsoft, Bob Blakley, IBM, Allen Brown,
1463 Microsoft, Joel Farrell, IBM, Mark Hayes, VeriSign, Kelvin Lawrence, IBM, Scott Konersmann,
1464 Microsoft, David Melgar, IBM, Dan Simon, Microsoft, Wayne Vicknair, IBM.

17 References

- 1465
- 1466 **[DIGSIG]** Informational RFC 2828, "[Internet Security Glossary](#)," May 2000.
- 1467 **[Kerberos]** J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," [RFC 1510](#), September 1993, <http://www.ietf.org/rfc/rfc1510.txt> .
- 1468
- 1469 **[KEYWORDS]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," [RFC 2119](#), Harvard University, March 1997
- 1470
- 1471 **[SHA-1]** FIPS PUB 180-1. Secure Hash Standard. U.S. Department of
- 1472 Commerce / National Institute of Standards and Technology.
- 1473 <http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>
- 1474 **[SOAP11]** W3C Note, "[SOAP: Simple Object Access Protocol 1.1](#)," 08 May 2000.
- 1475 **[SOAP12]** W3C Working Draft, "SOAP Version 1.2 Part 1: Messaging Framework",
- 1476 26 June 2002
- 1477 **[SOAP-SEC]** W3C Note, "[SOAP Security Extensions: Digital Signature](#)," 06 February
- 1478 2001.
- 1479 **[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.
- 1480
- 1481
- 1482 **[WS-Security]** "[Web Services Security Language](#)", IBM, Microsoft, VeriSign, April 2002.
- 1483 "[WS-Security Addendum](#)", IBM, Microsoft, VeriSign, August 2002.
- 1484 "[WS-Security XML Tokens](#)", IBM, Microsoft, VeriSign, August 2002.
- 1485 **[XML-C14N]** W3C Recommendation, "[Canonical XML Version 1.0](#)," 15 March 2001
- 1486 **[EXC-C14N]** W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8
- 1487 July 2002.
- 1488 **[XML-Encrypt]** W3C Working Draft, "[XML Encryption Syntax and Processing](#)," 04 March
- 1489 2002
- 1490 W3C Recommendation, "Decryption Transform for XML Signature", 10
- 1491 December 2002.
- 1492 **[XML-ns]** W3C Recommendation, "[Namespaces in XML](#)," 14 January 1999.
- 1493 **[XML-Schema]** W3C Recommendation, "[XML Schema Part 1: Structures](#)," 2 May 2001.
- 1494 W3C Recommendation, "[XML Schema Part 2: Datatypes](#)," 2 May 2001.
- 1495 **[XML Signature]** W3C Recommendation, "[XML Signature Syntax and Processing](#)," 12
- 1496 February 2002.
- 1497 **[X509]** S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified
- 1498 Certificates Profile,"

1499		http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-1
1500		
1501	[XPath]	W3C Recommendation, " XML Path Language ", 16 November 1999
1502	[WSS-SAML]	OASIS Working Draft 06, " Web Services Security SAML Token Profile ",
1503		21 February 2003
1504	[WSS-XrML]	OASIS Working Draft 03, " Web Services Security XrML Token Profile ",
1505		30 January 2003
1506	[WSS-X509]	OASIS Working Draft 03, " Web Services Security X509 Profile ", 30
1507		January 2003
1508	[WSS-Kerberos]	OASIS Working Draft 03, " Web Services Security Kerberos Profile ", 30
1509		January 2003
1510	[WSS-Username]	OASIS Working Draft 02, " Web Services Security UsernameToken Profile ",
1511		23 February 2003
1512	[WSS-XCBF]	OASIS Working Draft 1.1, " Web Services Security XCBF Token Profile ",
1513		30 March 2003
1514	[XPointer]	"XML Pointer Language (XPointer) Version 1.0, Candidate
1515		Recommendation", DeRose, Maler, Daniel, 11 September 2001.

1516

Appendix A: Utility Elements and Attributes

1517 This specification defines several elements, attributes, and attribute groups which can be re-used
1518 by other specifications. This appendix provides an overview of these *utility* components. It
1519 should be noted that the detailed descriptions are provided in the specification and this appendix
1520 will reference these sections as well as calling out other aspects not documented in the
1521 specification.

1522 A.1. Identification Attribute

1523 There are many situations where elements within **SOAP** messages need to be referenced. For
1524 example, when signing a SOAP message, selected elements are included in the signature. [XML](#)
1525 [Schema Part 2](#) provides several built-in data types that may be used for identifying and
1526 referencing elements, but their use requires that consumers of the SOAP message either to have
1527 or be able to obtain the schemas where the identity or reference mechanisms are defined. In
1528 some circumstances, for example, intermediaries, this can be problematic and not desirable.

1529 Consequently a mechanism is required for identifying and referencing elements, based on the
1530 SOAP foundation, which does not rely upon complete schema knowledge of the context in which
1531 an element is used. This functionality can be integrated into SOAP processors so that elements
1532 can be identified and referred to without dynamic schema discovery and processing.

1533 This specification specifies a namespace-qualified global attribute for identifying an element
1534 which can be applied to any element that either allows arbitrary attributes or specifically allows
1535 this attribute. This is a general purpose mechanism which can be re-used as needed.

1536 A detailed description can be found in [Section 4.0 ID References](#).

1537 A.2. Timestamp Elements

1538 The specification defines XML elements which may be used to express timestamp information
1539 such as creation, expiration, and receipt. While defined in the context of messages, these
1540 elements can be re-used wherever these sorts of time statements need to be made.

1541 The elements in this specification are defined and illustrated using time references in terms of the
1542 *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this
1543 type for interoperability. It is further RECOMMENDED that all references be in UTC time for
1544 increased interoperability. If, however, other time types are used, then the *ValueType* attribute
1545 MUST be specified to indicate the data type of the time format.

1546 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.

1547 A detailed description can be found in Section [10 Message Timestamp](#).

1548 **A.3. General Schema Types**

1549 The schema for the utility aspects of this specification also defines some general purpose
 1550 schema elements. While these elements are defined in this schema for use with this
 1551 specification, they are general purpose definitions that may be used by other specifications as
 1552 well.

1553 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.

1554

1555 Appendix B: SecurityTokenReference Model

1556 This appendix provides a non-normative overview of the usage and processing models for the
1557 <wsse:SecurityTokenReference> element.

1558 There are several motivations for introducing the <wsse:SecurityTokenReference>
1559 element:

1560 The XML Signature reference mechanisms are focused on "key" references rather than general
1561 token references.

1562 The XML Signature reference mechanisms utilize a fairly closed schema which limits the
1563 extensibility that can be applied.

1564 There are additional types of general reference mechanisms that are needed, but are not covered
1565 by XML Signature.

1566 There are scenarios where a reference may occur outside of an XML Signature and the XML
1567 Signature schema is not appropriate or desired.

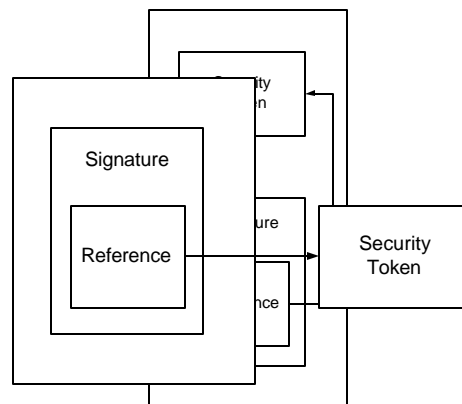
1568 The XML Signature references may include aspects (e.g. transforms) that may not apply to all
1569 references.

1570

1571 The following use cases drive the above motivations:

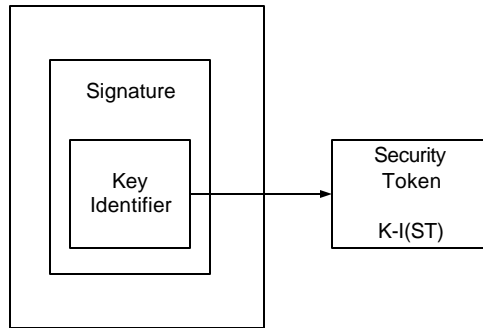
1572 **Local Reference** – A security token, that is included in the message in the <wsse:Security>
1573 header, is associated with an XML Signature. The figure below illustrates this:

1574



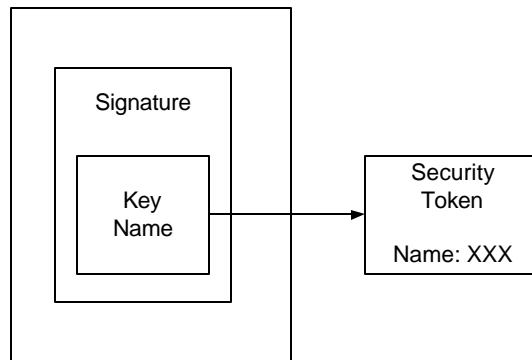
1575 **Remote Reference** – A security token, that is not included in the message but may be available
1576 at a specific URI, is associated with an XML Signature. The figure below illustrates this:

1577

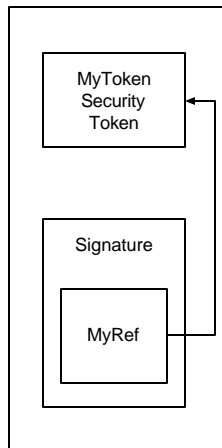


1578 **Key Identifier** – A security token, which is associated with an XML Signature and identified using
 1579 a known value that is the result of a well-known function of the security token (defined by the
 1580 token format or profile). The figure below illustrates this where the token is located externally:
 1581

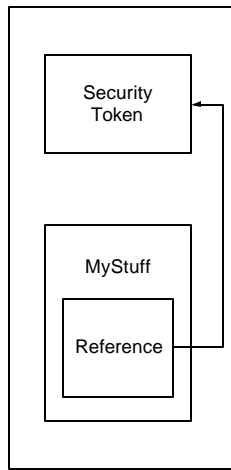
1582 **Key Name** – A security token is associated with an XML Signature and identified using a known
 1583 value that represents a "name" assertion within the security token (defined by the token format or
 1584 profile). The figure below illustrates this where the token is located externally:



1585 **Format-Specific References** – A security token is associated with an XML Signature and
 1586



1587 identified using a mechanism specific to the token (rather than the general mechanisms



1588 described above). The figure below illustrates this:

1589

1590

1591 **Non-Signature References** – A message may contain XML that does not represent an XML
 1592 signature, but may reference a security token (which may or may not be included in the
 1593 message). The figure below illustrates this:

1594

1595

1596 All conformant implementations **MUST** be able to process the
 1597 `<wsse:SecurityTokenReference>` element. However, they are not required to support all of
 1598 the different types of references.

1599 The reference **MAY** include a *ValueType* attribute which provides a "hint" for the type of desired
 1600 token.

1601 If multiple sub-elements are specified, together they describe the reference for the token.

1602 There are several challenges that implementations face when trying to interoperate:

1603 **ID References** – The underlying XML referencing mechanism using the XML base type of ID
 1604 provides a simple straightforward XML element reference. However, because this is an XML
 1605 type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
 1606 requires the recipient to *understand* the schema. This may be an expensive task and in the
 1607 general case impossible as there is no way to know the "schema location" for a specific
 1608 namespace URI.

1609 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
 1610 references are, by definition, unique by XML. However, other mechanisms such as "principal
 1611 name" are not required to be unique and therefore such references may be unique.

1612 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
 1613 information about the "key" used in the signature. For token references within signatures, it is
 1614 **RECOMMENDED** that the `<wsse:SecurityTokenReference>` be placed within the
 1615 `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
 1616 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
 1617 Message Security or its profiles are preferred over the mechanisms in XML Signature.

1618 The following provides additional details on the specific reference mechanisms defined in WSS:
 1619 SOAP Message Security:

1620 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
 1621 the security token. If only the fragment is specified, then it references the security token within
 1622 the document whose *wsu:Id* matches the fragment. For non-fragment URIs, the reference is to

1623 a [potentially external] security token identified using a URI. There are no implied semantics
1624 around the processing of the URI.

1625 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
1626 by specifying a known value (identifier) for the token, which is determined by applying a special
1627 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
1628 specific security token but requires a profile or token-specific function to be specified. The
1629 *ValueType* attribute provide a *hint* as to the desired token type. The *EncodingType* attribute
1630 specifies how the unique value (identifier) is encoded. For example, a hash value may be
1631 encoded using base 64 encoding (the default).

1632 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
1633 specific value that is used to *match* identity assertion within the security token. This is a subset
1634 match and may result in multiple security tokens that match the specified name. While XML
1635 Signature doesn't imply formatting semantics, WSS: SOAP Message Security RECOMMENDS
1636 that X.509 names be specified.

1637 It is expected that, where appropriate, profiles define if and how the reference mechanisms map
1638 to the specific token profile. Specifically, the profile should answer the following questions:

1639 What types of references can be used?
1640 How "Key Name" references map (if at all)?
1641 How "Key Identifier" references map (if at all)?
1642 Any additional profile or format-specific references?

1643
1644

1645

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

1646

Appendix D: Notices

1647

1648 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1649 that might be claimed to pertain to the implementation or use of the technology described in this
1650 document or the extent to which any license under such rights might or might not be available;
1651 neither does it represent that it has made any effort to identify any such rights. Information on
1652 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1653 website. Copies of claims of rights made available for publication and any assurances of licenses
1654 to be made available, or the result of an attempt made to obtain a general license or permission
1655 for the use of such proprietary rights by implementers or users of this specification, can be
1656 obtained from the OASIS Executive Director.

1657 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1658 applications, or other proprietary rights which may cover technology that may be required to
1659 implement this specification. Please address the information to the OASIS Executive Director.

1660 Copyright © OASIS Open 2002. *All Rights Reserved.*

1661 This document and translations of it may be copied and furnished to others, and derivative works
1662 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1663 published and distributed, in whole or in part, without restriction of any kind, provided that the
1664 above copyright notice and this paragraph are included on all such copies and derivative works.
1665 However, this document itself does not be modified in any way, such as by removing the
1666 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1667 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1668 Property Rights document must be followed, or as required to translate it into languages other
1669 than English.

1670 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1671 successors or assigns.

1672 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1673 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1674 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1675 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1676 PARTICULAR PURPOSE.

1677