

## Web Services Security Core Specification

Formatted: Border: Top: (Single solid line, Auto, 0.5 pt Line width, From text: 2 pt Border spacing: )

Working Draft 07, 11 December 2002

Deleted: 6

Deleted: 08

### Document identifier:

WSS-Core-06v

### Location:

TBD

### Editors:

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

### Contributors:

TBD – Revise this list to include WSS TC contributors

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

### Abstract:

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

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

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

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 .....	5
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 .....	9
52	3.1	Message Security Model.....	9
53	3.2	Message Protection .....	9
54	3.3	Invalid or Missing Claims .....	9
55	3.4	Example .....	10
56	4	ID References .....	12
57	4.1	Id Attribute .....	12
58	4.2	Id Schema .....	12
59	5	Security Header.....	14
60	6	Security Tokens .....	16
61	6.1	User Name Tokens .....	16
62	6.1.1	Usernames and Passwords.....	16
63	6.2	Binary Security Tokens .....	18
64	6.2.1	Attaching Security Tokens.....	18
65	6.2.2	Processing Rules .....	18
66	6.2.3	Encoding Binary Security Tokens .....	18
67	6.3	XML Tokens .....	20
68	6.3.1	Attaching Security Tokens.....	20
69	6.3.2	Identifying and Referencing Security Tokens .....	20
70	6.3.3	Subject Confirmation .....	20
71	6.3.4	Processing Rules .....	20
72	7	Token References .....	21
73	7.1	SecurityTokenReference Element .....	21
74	7.2	Direct References .....	22
75	7.3	Key Identifiers.....	23
76	7.4	ds:KeyInfo .....	23
77	7.5	Key Names .....	24
78	7.6	Token Reference Lookup Processing Order.....	24
79	8	Signatures .....	25
80	8.1	Algorithms .....	25
81	8.2	Signing Messages.....	26
82	8.3	Signature Validation.....	26
83	8.4	Example .....	26

84	9	Encryption.....	28
85	9.1	xenc:ReferenceList.....	28
86	9.2	xenc:EncryptedKey.....	29
87	9.3	xenc:EncryptedData.....	30
88	9.4	Processing Rules.....	30
89	9.4.1	Encryption.....	31
90	9.4.2	Decryption.....	31
91	9.5	Decryption Transformation.....	31
92	10	Message Timestamps.....	33
93	10.1	Model.....	33
94	10.2	Timestamp Elements.....	33
95	10.2.1	Creation.....	33
96	10.2.2	Expiration.....	34
97	10.3	Timestamp Header.....	34
98	10.4	TimestampTrace Header.....	36
99	11	Extended Example.....	38
100	12	Error Handling.....	41
101	13	Security Considerations.....	42
102	14	Privacy Considerations.....	44
103	15	Acknowledgements.....	45
104	16	References.....	46
105		Appendix A: Utility Elements and Attributes.....	48
106	A.1.	Identification Attribute.....	48
107	A.2.	Timestamp Elements.....	48
108	A.3.	General Schema Types.....	49
109		Appendix B: SecurityTokenReference Model.....	50
110		Appendix C: Revision History.....	54
111		Appendix D: Notices.....	55
112			

## 113 1 Introduction

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

Deleted: d

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

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

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

### 131 1.1 Goals and Requirements

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

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

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

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

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

#### 144 1.1.1 Requirements

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

- 147 • Multiple security token formats
- 148 • Multiple trust domains
- 149 • Multiple signature formats
- 150 • Multiple encryption technologies
- 151 • End-to-end message-level security and not just transport-level security

#### 152 1.1.2 Non-Goals

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

- 154 • Establishing a security context or authentication mechanisms.

- 155 • Key derivation.
- 156 • Advertisement and exchange of security policy.
- 157 • How trust is established or determined.
- 158

Deleted: Advertisement

## 2 Notations and Terminology

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

### 2.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

Namespace URIs (of the general form "some-URI") **represents** some application-dependent or context-dependent URI as defined in RFC2396.

Deleted: represent

In this document the style chosen when describing elements use is to XPath-like Notation. The XPath-like notation is declarative rather than procedural. Each pattern describes the types of nodes to match using a notation that indicates the hierarchical relationship between the nodes. For example, the pattern "/author" means find "author" elements contained in "root" element. The following operators and special characters are used in this document:

Formatted: Font: Arial, 10 pt

Formatted: Font: Arial, 10 pt

Formatted: Font: Arial, 10 pt

Formatted: Font: 10 pt

**/** - Child operator: selects immediate children of the left-side collection. When this path operator appears at the start of the pattern, it indicates that children should be selected from the root node.

Formatted: Definition term Char

**@** - Attribute: prefix for an attribute name

Formatted: Italic,i

**{any}** - Wildcard

Formatted: Definition term Char

Formatted: Definition term Char

Formatted: Italic,i

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

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

### 2.2 Namespaces

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

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

The following namespaces are used in this document:

Prefix	Namespace
S	<a href="http://www.w3.org/2001/12/soap-envelope">http://www.w3.org/2001/12/soap-envelope</a>
ds	<a href="http://www.w3.org/2000/09/xmldsig#">http://www.w3.org/2000/09/xmldsig#</a>
xenc	<a href="http://www.w3.org/2001/04/xmlenc#">http://www.w3.org/2001/04/xmlenc#</a>
wsse	<a href="http://schemas.xmlsoap.org/ws/2002/xx/secext">http://schemas.xmlsoap.org/ws/2002/xx/secext</a>

189 **2.3 Terminology**

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

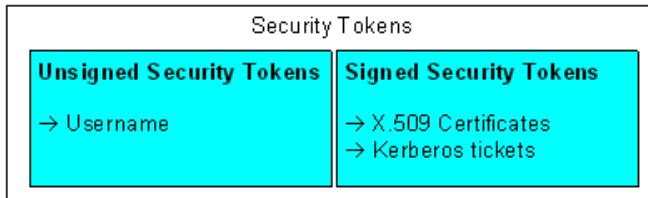
191 **Claim** – A *claim* is a declaration made by an entity (e.g. name, identity, key, group, privilege, capability, etc).

Deleted: client

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

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

196



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

Deleted: .

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

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

Deleted: with

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

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

Deleted: with

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

208 **Signature** - A *signature* is a cryptographic binding between a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved.

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

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

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

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

Deleted: Messgae

225



## 226 3 Message Protection Mechanisms

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

Deleted: When securing

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

### 232 3.1 Message Security Model

233 | This document specifies an abstract *message security model* in terms of [security tokens](#)  
234 | combined with digital [signatures](#) to protect and authenticate SOAP messages.

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

243 | Signatures are also used by message senders to demonstrate knowledge of the key claimed in a  
244 | security token and thus to authenticate or bind their identity (and any other claims occurring in the  
245 | security token) to the messages they create. A signature created by a message sender to  
246 | demonstrate knowledge of an authentication key is referred to as a [Proof-of-Possession](#) and may  
247 | serve as a message authenticator if the signature is performed over the message.

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

### 250 3.2 Message Protection

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

Deleted: .

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

259 | Message [confidentiality](#) leverages [XML Encryption](#) in conjunction with [security tokens](#) to keep  
260 | portions of a [SOAP](#) message [confidential](#). The encryption mechanisms are designed to support  
261 | additional encryption processes and operations by multiple [SOAP](#) roles.

262 | This document defines syntax and semantics of signatures within `<wsse:Security>` element.  
263 | This document also does not specify any signature appearing outside of `<wsse:Security>`  
264 | element, if any.

### 265 3.3 Invalid or Missing Claims

266 | The message recipient SHOULD reject a message with a signature determined to be invalid,  
267 | missing or unacceptable [claims](#) as it is an unauthorized (or malformed) message. This  
268 | specification provides a flexible way for the message sender to make a [claim](#) about the security  
269 | properties by associating zero or more [security tokens](#) with the message. An example of a

270 security **claim** is the identity of the sender; the sender can **claim** that he is Bob, known as an  
271 employee of some company, and therefore he has the right to send the message.

### 272 3.4 Example

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

```
281 (001) <?xml version="1.0" encoding="utf-8"?>
282 (002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
283       xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
284 (003)   <S:Header>
285 (004)     <wsse:Security
286           xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
287 (005)       <wsse:UsernameToken wsu:Id="MyID">
288 (006)         <wsse:Username>Zoe</wsse:Username>
289 (007)         <wsse:Nonce>FKJh...</wsse:Nonce>
290 (008)         <wsu:Created>2001-10-13T09:00:00Z</wsu:Created>
291 (009)       </wsse:UsernameToken>
292 (010)       <ds:Signature>
293 (011)         <ds:SignedInfo>
294 (012)           <ds:CanonicalizationMethod
295                   Algorithm=
296                     "http://www.w3.org/2001/10/xml-exc-c14n#" />
297 (013)           <ds:SignatureMethod
298                   Algorithm=
299                     "http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
300 (014)           <ds:Reference URI="#MsgBody">
301 (015)             <ds:DigestMethod
302                   Algorithm=
303                     "http://www.w3.org/2000/09/xmldsig#sha1" />
304 (016)             <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
305 (017)             </ds:Reference>
306 (018)           </ds:SignedInfo>
307 (019)           <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
308 (020)           <ds:KeyInfo>
309 (021)             <wsse:SecurityTokenReference>
310 (022)               <wsse:Reference URI="#MyID" />
311 (023)             </wsse:SecurityTokenReference>
312 (024)           </ds:KeyInfo>
313 (025)           </ds:Signature>
314 (026)         </wsse:Security>
315 (027)   </S:Header>
316 (028)   <S:Body wsu:Id="MsgBody">
317 (029)     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
318           QQQ
319         </tru:StockSymbol>
320 (030)   </S:Body>
321 (031) </S:Envelope>
```

Deleted: \

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

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

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

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

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

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

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

346 Lines (028) to (030) contain the *body* (payload) of the [SOAP](#) message.  
347

## 348 4 ID References

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

- 353 • ID attributes from XML Signature
- 354 • ID attributes from XML Encryption
- 355 • wsu:Id global attribute described below

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.

### 359 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.

### 374 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/2002/xx/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.

Deleted:

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 6 Security Tokens

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

### 460 6.1 User Name Tokens

#### 461 6.1.1 Usernames and Passwords

462 The `<wsse:UsernameToken>` element is introduced as a way of providing a username and  
463 optional password information. This element is optionally included in the `<wsse:Security>`  
464 header.

465 Within this element, a `<wsse:Password>` element MAY be specified. The password has an  
466 associated type – either `wsse:PasswordText` or `wsse:PasswordDigest`. The  
467 `wsse:PasswordText` is not limited to the actual password. Any password equivalent such as a  
468 derived password or S/KEY (one time password) can be used.

469 The `wsse:PasswordDigest` is defined as a base64-encoded SHA1 hash value of the UTF8-  
470 encoded password. However, unless this digested password is sent on a secured channel, the  
471 digest offers no real additional security than `wsse:PasswordText`.

472 To address this issue, two optional elements are introduced in the `<wsse:UsernameToken>`  
473 element: `<wsse:Nonce>` and `<wsu:Created>`. If either of these is present, they MUST be  
474 included in the digest value as follows:

```
475 PasswordDigest = SHA1 ( nonce + created + password )
```

476 That is, concatenate the nonce, creation timestamp, and the password (or shared secret or  
477 password equivalent) and include the digest of the combination. This helps obscure the  
478 password and offers a basis for preventing replay attacks. It is RECOMMENDED that timestamps  
479 and nonces be cached for a given period of time, as a guideline a value of five minutes can be  
480 used as a minimum to detect replays, and that timestamps older than that given period of time set  
481 be rejected.

482 Note that the nonce is hashed using the octet sequence of its decoded value while the timestamp  
483 is hashed using the octet sequence of its UTF8 encoding as specified in the contents of the  
484 element.

485 Note that password digests SHOULD NOT be used unless the plain text password, secret, or  
486 password-equivalent is available to both the requestor and the recipient.

487 The following illustrates the syntax of this element:

```
488 <wsse:UsernameToken wsu:Id="...">  
489   <wsse:Username>...</wsse:Username>  
490   <wsse:Password Type="...">...</wsse:Password>  
491   <wsse:Nonce EncodingType="...">...</wsse:Nonce>  
492   <wsu:Created>...</wsu:Created>  
493 </wsse:UsernameToken>
```

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

495 */wsse:UsernameToken*

496 This element is used for sending basic authentication information.

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

498 A string label for this security token.

499 */wsse:UsernameToken/Username*



500 This required element specifies the username of the authenticated or the party to be  
501 authenticated.

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

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

505 */wsse:UsernameToken/Password*

506 This optional element provides password information. It is RECOMMENDED that this  
507 element only be passed when a secure transport is being used.

508 */wsse:UsernameToken/Password/@Type*

509 This optional attribute specifies the type of password being provided. The following table  
510 identifies the pre-defined types:

Value	Description
wsse:PasswordText (default)	The actual password for the username or derived password or S/KEY .
wsse:PasswordDigest	The digest of the password for the username using the algorithm described above.

511 */wsse:UsernameToken/Password/@{any}*

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

514 */wsse:UsernameToken//wsse:Nonce*

515 This optional element specifies a cryptographically random nonce.

516 */wsse:UsernameToken//wsse:Nonce/@EncodingType*

517 This optional attribute specifies the encoding type of the nonce (see definition of  
518 `<wsse:BinarySecurityToken>` for valid values). If this attribute isn't specified then  
519 the default of Base64 encoding is used.

520 */wsse:UsernameToken//wsu:Created*

521 This optional element specifies the time (according to the originator) at which the  
522 password digest was created.

523 */wsse:UsernameToken/{any}*

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

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

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

529 All compliant implementations MUST be able to process a `<wsse:UsernameToken>` element.  
530 The following illustrates the use of this element (note that in this example the password is sent in  
531 clear text and the message should therefore be sent over a confidential channel:

```

532 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
533           xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
534   <S:Header>
535     ...
536     <wsse:Security>
537       <wsse:UsernameToken >
538         <wsse:Username>Zoe</wsse:Username>
539         <wsse:Password>ILoveDogs</wsse:Password>
540       </wsse:UsernameToken>
541     </wsse:Security>

```

```
542     ...
543     </S:Header>
544     ...
545 </S:Envelope>
```

546 The following example illustrates a hashed password using both a nonce and a timestamp with  
547 the password hashed:

```
548 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
549           xmlns:wssse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
550   <S:Header>
551     ...
552     <wsse:Security>
553       <wsse:UsernameToken
554         xmlns:wssse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
555         xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
556         <wsse:Username>NNK</wsse:Username>
557         <wsse:Password Type="wsse:PasswordDigest">
558           FEeR...</wsse:Password>
559         <wsse:Nonce>FKJh...</wsse:Nonce>
560         <wsu:Created>2001-10-13T09:00:00Z </wsu:Created>
561       </wsse:UsernameToken>
562     </wsse:Security>
563     ...
564   </S:Header>
565   ...
566 </S:Envelope>
```

## 567 6.2 Binary Security Tokens

### 568 6.2.1 Attaching Security Tokens

569 For binary-formatted security tokens, this specification provides a

570 <wsse:BinarySecurityToken> element that can be included in the <wsse:Security>  
571 header block.

Deleted: .

### 572 6.2.2 Processing Rules

573 This specification describes the processing rules for using and processing [XML Signature](#) and  
574 [XML Encryption](#). These rules MUST be followed when using any type of security token including  
575 XML-based tokens. Note that this does NOT mean that binary security tokens MUST be signed  
576 or encrypted – only that if signature or encryption is used in conjunction with binary security  
577 tokens, they MUST be used in a way that conforms to the processing rules defined by this  
578 specification.

### 579 6.2.3 Encoding Binary Security Tokens

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

584 The <wsse:BinarySecurityToken> element defines two attributes that are used to interpret  
585 it. The `ValueType` attribute indicates what the security token is, for example, a [Kerberos](#) ticket.  
586 The `EncodingType` tells how the security token is encoded, for example `Base64Binary`.

587 The following is an overview of the syntax:

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

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

592 `/wsse:BinarySecurityToken`

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

594 `/wsse:BinarySecurityToken/@wsu:Id`

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

596 `/wsse:BinarySecurityToken/@ValueType`

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

602 `/wsse:BinarySecurityToken/@EncodingType`

603 The `EncodingType` attribute is used to indicate, using a QName, the encoding format of the binary data (e.g., `wsse:Base64Binary`). A new attribute is introduced, as there are currently issues that make derivations of mixed simple and complex types difficult within [XML Schema](#). The `EncodingType` attribute is interpreted to indicate the encoding format of the element. The following encoding formats are pre-defined:

QName	Description
<code>wsse:Base64Binary</code>	<a href="#">XML Schema</a> base 64 encoding

608 `/wsse:BinarySecurityToken/@{any}`

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

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

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

618 `<wsse:BinarySecurityToken>` element if this token does not carry the validating key (and consequently it is not cryptographically bound to the [signature](#)). For example, if we wanted to sign the previous example, we need to include the consumed namespace definitions.

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

```
624 <wsse:BinarySecurityToken
625   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext "
626   wsu:Id="myToken"
627   ValueType="x:MyType" xmlns:x="http://www.fabrikam123.com/x"
628   EncodingType="wsse:Base64Binary">
629   MIIIEZzCCA9CgAwIBAgIQEmtJZc0...
630 </wsse:BinarySecurityToken>
```

## 631 **6.3 XML Tokens**

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

### 635 **6.3.1 Attaching Security Tokens**

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

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

### 641 **6.3.2 Identifying and Referencing Security Tokens**

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

### 647 **6.3.3 Subject Confirmation**

648 This specification does not dictate if and how subject confirmation must be done, however, it does  
649 define how signatures can be used and associated with security tokens (by referencing them in  
650 the signature) as a form of Proof-of-~~Possession~~.

Deleted: Possession

### 651 **6.3.4 Processing Rules**

652 This specification describes the processing rules for using and processing XML Signature and  
653 XML Encryption. These rules MUST be followed when using any type of security token including  
654 XML-based tokens. Note that this does NOT mean that XML-based tokens MUST be signed or  
655 encrypted – only that if signature or encryption is used in conjunction with XML-based tokens,  
656 they MUST be used in a way that conforms to the processing rules defined by this specification.

## 657 7 Token References

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

### 659 7.1 SecurityTokenReference Element

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

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

667 The following illustrates the syntax of this element:

```
668 <wsse:SecurityTokenReference wsu:Id="..." >  
669 ...  
670 </wsse:SecurityTokenReference>
```

671 The following describes the elements defined above:

672 `<wsse:SecurityTokenReference`

Deleted: /

673 This element provides a reference to a security token.

674 `<wsse:SecurityTokenReference/@wsu:Id`

Deleted: /

675 A string label for this [security token](#) reference.

676 `<wsse:SecurityTokenReference/@wsse:Usage`

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

Formatted: Code Embedded, ce

QName	Description
<a href="#">wsse:UsageBind</a> (default)	<a href="#">This usage is for general binding of assertions. When used within a signature, the assertions of the referenced security token apply to the signed data.</a>

680

681 `<wsse:SecurityTokenReference/{any}`

Deleted: /

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

684 `<wsse:SecurityTokenReference/@{any}`

Deleted: /

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

687 All compliant implementations MUST be able to process a

688 `<wsse:SecurityTokenReference>` element.

689 This element can also be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to  
690 retrieve the key information from a security token placed somewhere else. In particular, it is  
691 RECOMMENDED, when using [XML Signature](#) and [XML Encryption](#), that a

Deleted: The following illustrates the use of this element:¶  
<wsse:SecurityTokenReference  
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"> ¶  
 <wsse:Reference ¶  
 URI="http://www.fabrikam123.com/tokens/Zoe#X509token"/>¶  
</wsse:SecurityTokenReference>¶

692 <wsse:SecurityTokenReference> element be placed inside a <ds:KeyInfo> to reference  
693 the **security token** used for the signature or encryption.

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

Formatted: Space Before: 4 pt,  
After: 4 pt, No bullets or numbering

701 The following list provides a list of the specific reference mechanisms defined in WS-Security in  
702 preferred order (i.e., most specific to least specific):

- 703 • **Direct References** – This allows references to included tokens using URI fragments and  
704 external tokens using full URIs.
- 705 • **Key Identifiers** – This allows tokens to be referenced using an opaque value that  
706 represents the token (defined by token type/profile).
- 707 • **Key Names** – This allows tokens to be referenced using a string that matches an identity  
708 assertion within the security token. This is a subset match and may result in multiple  
709 security tokens that match the specified name.

Formatted: Bullets and Numbering

Deleted: ¶

## 710 7.2 Direct References

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

713 The following illustrates the syntax of this element:

```
714 <wsse:SecurityTokenReference wsu:Id="..." >  
715 <wsse:Reference URI="..." ValueType="..." />  
716 </wsse:SecurityTokenReference>
```

717 The following describes the elements defined above:

718 ~~wsse:SecurityTokenReference/Reference~~

Deleted: /

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

720 ~~wsse:SecurityTokenReference/Reference/@URI~~

Deleted: /

721 This optional attribute specifies an **abstract** URI for where to find a security token.

722 ~~wsse:SecurityTokenReference/Reference/@ValueType~~

Deleted: /

723 This optional attribute specifies a QName that is used to identify the *type* of token being  
724 referenced (see <wsse:BinarySecurityToken>). This specification does not define  
725 any processing rules around the usage of this attribute, however, specifications for  
726 individual token types MAY define specific processing rules and semantics around the  
727 value of the URI and how it SHALL be interpreted. If this attribute is not present, the URI  
728 SHALL be processed as a normal URI.

729 ~~wsse:SecurityTokenReference/Reference/{any}~~

Deleted: /

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

732 ~~wsse:SecurityTokenReference/Reference/@{any}~~

Deleted: /

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

735 The following illustrates the use of this element:

```
736 <wsse:SecurityTokenReference  
737 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext ">  
738 <wsse:Reference  
739 URI="http://www.fabrikaml23.com/tokens/Zoe#X509token" />
```

740 </wsse:SecurityTokenReference>

### 741 7.3 Key Identifiers

742 If a direct reference is not possible, then it is RECOMMENDED to use a key identifier to  
743 specify/reference a security token instead of a key name. The <wsse:KeyIdentifier>  
744 element SHALL be placed in the <wsse:SecurityTokenReference> element to reference a  
745 token using an identifier. This element SHOULD be used for all key identifiers.

746 The processing model assumes that the key identifier for a security token is constant.  
747 Consequently, processing a key identifier is simply looking for a security token whose key  
748 identifier matches a given specified constant.

749 The following is an overview of the syntax:

```
750 <wsse:SecurityTokenReference>
751   <wsse:KeyIdentifier wsu:Id="..."
752                       ValueType="..."
753                       EncodingType="...">
754     ...
755   </wsse:KeyIdentifier>
756 </wsse:SecurityTokenReference>
```

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

758 /wsse:SecurityTokenReference/KeyIdentifier Deleted:

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

760 /wsse:SecurityTokenReference/KeyIdentifier/@wsu:Id Deleted:

761 An optional string label for this identifier.

762 /wsse:SecurityTokenReference/KeyIdentifier/@ValueType Deleted:

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

767 /wsse:SecurityTokenReference/KeyIdentifier/@EncodingType Deleted:

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

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

771 /wsse:SecurityTokenReference/KeyIdentifier/{any} Deleted:

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

### 774 7.4 ds:KeyInfo

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

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

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

## 784 7.5 Key Names

785 | It is strongly RECOMMENDED to use key identifiers. However, if key names are used, then it is  
786 strongly RECOMMENDED that <ds:KeyName> elements conform to the attribute names in  
787 section 2.3 of RFC 2253 (this is recommended by XML Signature for <X509SubjectName>) for  
788 interoperability.

789 Additionally, defined are the following convention for e-mail addresses, which SHOULD conform  
790 to RFC 822:

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

## 792 7.6 Token Reference Lookup Processing Order

793 There are a number of mechanisms described in [XML Signature](#) and this specification  
794 for referencing security tokens. To resolve possible ambiguities when more than one  
795 of these reference constructs is included in a single KeyInfo element, the following  
796 processing order SHOULD be used:

- 797 1. Resolve any <wsse:Reference> elements (specified within  
798 <wsse:SecurityTokenReference>).
- 799 2. Resolve any <wsse:KeyIdentifier> elements (specified within  
800 <wsse:SecurityTokenReference>).
- 801 3. Resolve any <ds:KeyName> elements.
- 802 4. Resolve any other <ds:KeyInfo> elements.

803 The processing stops as soon as one key has been located.



## 804 8 Signatures

805 Message senders may want to enable message recipients to determine whether a message was  
806 altered in transit and to verify that a message was sent by the possessor of a particular [security](#)  
807 [token](#).

808 An XML Digital Signature can bind claims with a SOAP message body and/or headers by  
809 associating those claims with a signing key. Accepting the binding and using the claims is at the  
810 discretion of the relying party. Placing claims in one or more `<SecurityToken>` elements that  
811 also convey the signing keys is the mechanism to create the binding of the claims. Each of these  
812 SecurityToken elements must be referenced with a `<SecurityTokenReference>` in the  
813 `<ds:KeyInfo>` element in the signature. The `<SecurityToken>` elements can be signed, or  
814 not, depending on the relying party trust model and other requirements.

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

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

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

### 832 8.1 Algorithms

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

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

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

837 The [Exclusive XML Canonicalization](#) algorithm addresses the pitfalls of general canonicalization  
838 that can occur from *leaky* namespaces with pre-existing signatures.

839 Finally, if a sender wishes to sign a message before encryption, they should use the [Decryption](#)  
840 [Transformation for XML Signature](#).

**Deleted:** The validation of an XML signature that uses a SecurityTokenReference to identify the key that may be used to validate the signature, supports the confirmation (by the relying party/recipient) of any other claims made within the referenced token (most notably the identity bound to the key) to the signature author (that is, if the relying party trusts the authority responsible for the claims in the referenced token).

**Formatted:** Code Embedded, ce

**Formatted:** Code Embedded, ce

**Formatted:** Code Embedded, ce

**Formatted:** Code Embedded, ce

## 8.2 Signing Messages

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

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

1. The application MUST be capable of processing the required elements defined in the [XML Signature](#) specification.
2. To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element conforming to the [XML Signature](#) specification SHOULD be prepended to the existing content of the `<wsse:Security>` header block. All the `<ds:Reference>` elements contained in the signature SHOULD refer to a resource within the enclosing [SOAP](#) envelope, or in an attachment.

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

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

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

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

## 8.3 Signature Validation

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

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

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

## 8.4 Example

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

```
<?xml version="1.0" encoding="utf-8"?>
```

```

887 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
888     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
889     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
890     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
891   <S:Header>
892     <wsse:Security>
893       <wsse:BinarySecurityToken
894         ValueType="wsse:X509v3"
895         EncodingType="wsse:Base64Binary"
896         wsu:Id="X509Token">
897         MIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
898       </wsse:BinarySecurityToken>
899       <ds:Signature>
900         <ds:SignedInfo>
901           <ds:CanonicalizationMethod Algorithm=
902             "http://www.w3.org/2001/10/xml-exc-c14n#" />
903           <ds:SignatureMethod Algorithm=
904             "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
905           <ds:Reference URI="#myBody">
906             <ds:Transforms>
907               <ds:Transform Algorithm=
908                 "http://www.w3.org/2001/10/xml-exc-c14n#" />
909             </ds:Transforms>
910             <ds:DigestMethod Algorithm=
911               "http://www.w3.org/2000/09/xmldsig#sha1" />
912             <ds:DigestValue>EULddytSol...</ds:DigestValue>
913           </ds:Reference>
914         </ds:SignedInfo>
915         <ds:SignatureValue>
916         BL8jdfToEbl1/vXcMZNNjPOV...
917       </ds:SignatureValue>
918       <ds:KeyInfo>
919         <wsse:SecurityTokenReference>
920           <wsse:Reference URI="#X509Token" />
921         </wsse:SecurityTokenReference>
922       </ds:KeyInfo>
923     </ds:Signature>
924   </wsse:Security>
925 </S:Header>
926 <S:Body wsu:Id="myBody" >
927   <tru:StockSymbol xmlns:tru="http://www.fabrikaml23.com/payloads">
928     QQQ
929   </tru:StockSymbol>
930 </S:Body>
931 </S:Envelope>

```

## 9 Encryption

932

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

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

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

Deleted: add

Deleted:

Deleted:

Deleted: hav €

Deleted: ¶

Formatted: Bullets and Numbering

### 9.1 `xenc:ReferenceList`

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

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

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

```
964 <S:Envelope  
965   xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
966   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
967   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"  
968   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">  
969   <S:Header>  
970     <wsse:Security>  
971       <xenc:ReferenceList>  
972         <xenc:DataReference URI="#bodyID"/>  
973       </xenc:ReferenceList>  
974     </wsse:Security>  
975   </S:Header>  
976   <S:Body>  
977     <xenc:EncryptedData Id="bodyID">  
978       <ds:KeyInfo>  
979         <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>  
980       </ds:KeyInfo>
```

```

981     <xenc:CipherData>
982         <xenc:CipherValue>...</xenc:CipherValue>
983     </xenc:CipherData>
984     </xenc:EncryptedData>
985 </S:Body>
986 </S:Envelope>

```

## 987 9.2 xenc:EncryptedKey

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

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

```

999 <S:Envelope
1000     xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1001     xmlns:ds="http://www.w3.org/2000/09/xmlsig#"
1002     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
1003     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1004     <S:Header>
1005         <wsse:Security>
1006             <xenc:EncryptedKey>
1007                 <xenc:EncryptionMethod Algorithm="..."/>
1008                 <ds:KeyInfo>
1009                     <wsse:SecurityTokenReference>
1010                         <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1011                             ValueType="wsse:X509v3">MIGfMa0GCSq...
1012                         </wsse:KeyIdentifier>
1013                     </wsse:SecurityTokenReference>
1014                 </ds:KeyInfo>
1015                 <xenc:CipherData>
1016                     <xenc:CipherValue>...</xenc:CipherValue>
1017                 </xenc:CipherData>
1018                 <xenc:ReferenceList>
1019                     <xenc:DataReference URI="#bodyID"/>
1020                 </xenc:ReferenceList>
1021             </xenc:EncryptedKey>
1022         </wsse:Security>
1023     </S:Header>
1024     <S:Body>
1025         <xenc:EncryptedData Id="bodyID">
1026             <xenc:CipherData>
1027                 <xenc:CipherValue>...</xenc:CipherValue>
1028             </xenc:CipherData>
1029         </xenc:EncryptedData>
1030     </S:Body>
1031 </S:Envelope>

```

1032 While XML Encryption specifies that <xenc:EncryptedKey> elements MAY be specified in  
1033 <xenc:EncryptedData> elements, this specification strongly RECOMMENDS that  
1034 <xenc:EncryptedKey> elements be placed in the <wsse:Security> header.

### 1035 9.3 xenc:EncryptedData

1036 In some cases security-related information is provided in a purely encrypted form or non-XML  
1037 attachments MAY be encrypted. The <xenc:EncryptedData> element from [XML Encryption](#)  
1038 SHALL be used for these scenarios. For each part of the encrypted attachment, one encryption  
1039 step is needed; that is, for each attachment to be encrypted, one <xenc:EncryptedData> sub-  
1040 element MUST be added with the following rules (note that steps 2-4 applies only if MIME types  
1041 are being used for attachments).

- 1042 1. The contents of the attachment MUST be replaced by the encrypted octet string.
- 1043 2. The replaced MIME part MUST have the media type `application/octet-stream`.
- 1044 3. The original media type of the attachment MUST be declared in the `MimeType` attribute  
1045 of the <xenc:EncryptedData> element.
- 1046 4. The encrypted MIME part MUST be referenced by an <xenc:CipherReference>  
1047 element with a URI that points to the MIME part with `cid:` as the scheme component of  
1048 the URI.

1049 The following illustrates the use of this element to indicate an encrypted attachment:

```
1050 <S:Envelope  
1051   xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
1052   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"  
1053   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"  
1054   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">  
1055   <S:Header>  
1056     <wsse:Security>  
1057       <xenc:EncryptedData MimeType="image/png">  
1058         <ds:KeyInfo>  
1059           <wsse:SecurityTokenReference>  
1060             <xenc:EncryptionMethod Algorithm="..."/>  
1061             <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"  
1062               ValueType="wsse:X509v3">MIGfMa0GCSq...  
1063           </wsse:KeyIdentifier>  
1064         </wsse:SecurityTokenReference>  
1065       </ds:KeyInfo>  
1066       <xenc:CipherData>  
1067         <xenc:CipherReference URI="cid:image"/>  
1068       </xenc:CipherData>  
1069     </xenc:EncryptedData>  
1070   </wsse:Security>  
1071 </S:Header>  
1072 <S:Body> </S:Body>  
1073 </S:Envelope>
```

### 1074 9.4 Processing Rules

1075 Encrypted parts or attachments to the [SOAP](#) message using one of the sub-elements defined  
1076 above MUST be in compliance with the [XML Encryption](#) specification. An encrypted [SOAP](#)  
1077 envelope MUST still be a valid [SOAP](#) envelope. The message creator MUST NOT encrypt the  
1078 <S:Envelope>, <S:Header>, or <S:Body> elements but MAY encrypt child elements of  
1079 either the <S:Header> and <S:Body> elements. Multiple steps of encryption MAY be added  
1080 into a single <Security> header block if they are targeted for the same recipient.

1081 When an element or element content inside a [SOAP](#) envelope (e.g. of the contents of <S:Body>)  
1082 is to be encrypted, it MUST be replaced by an <xenc:EncryptedData>, according to [XML](#)  
1083 [Encryption](#) and it SHOULD be referenced from the <xenc:ReferenceList> element created  
1084 by this encryption step. This specification allows placing the encrypted octet stream in an  
1085 attachment. For example, if an <xenc:EncryptedData> element in an <S:Body> element has  
1086 <xenc:CipherReference> that refers to an attachment, then the decrypted octet stream

Deleted:

1087 SHALL replace the <xenc:EncryptedData>. However, if the <xenc:EncryptedData>  
1088 element is located in the <Security> header block and it refers to an attachment, then the  
1089 decrypted octet stream MUST replace the encrypted octet stream in the attachment.

## 1090 9.4.1 Encryption

1091 The general steps (non-normative) for creating an encrypted SOAP message in compliance with  
1092 this specification are listed below (note that use of <xenc:ReferenceList> is  
1093 RECOMMENDED).

- 1094 1. Create a new SOAP envelope.
- 1095 2. Create a <Security> header
- 1096 3. Create an <xenc:ReferenceList> sub-element, an <xenc:EncryptedKey> sub-  
1097 element, or an <xenc:EncryptedData> sub-element in the <Security> header  
1098 block (note that if the SOAP "role" and "mustUnderstand" attributes are different, then a  
1099 new header block may be necessary), depending on the type of encryption.
- 1100 4. Locate data items to be encrypted, i.e., XML elements, element contents within the target  
1101 SOAP envelope, and attachments.
- 1102 5. Encrypt the data items as follows: For each XML element or element content within the  
1103 target SOAP envelope, encrypt it according to the processing rules of the XML  
1104 Encryption specification. Each selected original element or element content MUST be  
1105 removed and replaced by the resulting <xenc:EncryptedData> element. For an  
1106 attachment, the contents MUST be replaced by encrypted cipher data as described in  
1107 section 9.3 Signature Validation.
- 1108 6. The optional <ds:KeyInfo> element in the <xenc:EncryptedData> element MAY  
1109 reference another <ds:KeyInfo> element. Note that if the encryption is based on an  
1110 attached security token, then a <SecurityTokenReference> element SHOULD be  
1111 added to the <ds:KeyInfo> element to facilitate locating it.
- 1112 7. Create an <xenc:DataReference> element referencing the generated  
1113 <xenc:EncryptedData> elements. Add the created <xenc:DataReference>  
1114 element to the <xenc:ReferenceList>.

## 1115 9.4.2 Decryption

1116 On receiving a SOAP envelope containing encryption header elements, for each encryption  
1117 header element the following general steps should be processed (non-normative):

- 1118 1. Locate the <xenc:EncryptedData> items to be decrypted (possibly using the  
1119 <xenc:ReferenceList>).
- 1120 2. Decrypt them as follows: For each element in the target SOAP envelope, decrypt it  
1121 according to the processing rules of the XML Encryption specification and the processing  
1122 rules listed above.
- 1123 3. If the decrypted data is part of an attachment and MIME types were used, then revise the  
1124 MIME type of the attachment to the original MIME type (if one exists).

1125 If the decryption fails for some reason, applications MAY report the failure to the sender using the  
1126 fault code defined in Section 12 Error Handling.

## 1127 9.5 Decryption Transformation

1128 The ordering semantics of the <wsse:Security> header are sufficient to determine if  
1129 signatures are over encrypted or unencrypted data. However, when a signature is included in  
1130 one <wsse:Security> header and the encryption data is in another <wsse:Security>  
1131 header, the proper processing order may not be apparent.

1132 If the sender wishes to sign a message that MAY subsequently be encrypted by an intermediary  
1133 then the sender MAY use the Decryption Transform for XML Signature to explicitly specify the  
1134 order of decryption.  
1135



## 1136 **10 Message Timestamps**

1137 It is often important for the recipient to be able to determine the *freshness* of a message. In some  
1138 cases, a message may be so *stale* that the recipient may decide to ignore it.

1139 This specification does not provide a mechanism for synchronizing time. The assumption is  
1140 either that the recipient is using a mechanism to synchronize time (e.g. NTP) or, more likely for  
1141 federated applications, that they are making assessments about time based on three factors:  
1142 creation time of the message, transmission checkpoints, and transmission delays and their local  
1143 time.

1144 To assist a recipient in making an assessment of staleness, a requestor may wish to indicate a  
1145 suggested expiration time after which the recipient should ignore the message. The specification  
1146 provides XML elements by which the requestor may express the expiration time of a message,  
1147 the requestor's clock time at the moment the message was created, checkpoint timestamps  
1148 (when an **SOAP** role received the message) along the communication path, and the delays  
1149 introduced by transmission and other factors subsequent to creation. The quality of the delays is  
1150 a function of how well they reflect the actual delays (e.g., how well they reflect transmission  
1151 delays).

1152 It should be noted that this is not a protocol for making assertions or determining when, or how  
1153 fast, a service produced or processed a message.

1154 This specification defines and illustrates time references in terms of the *dateTimeType* defined in  
1155 XML Schema. It is RECOMMENDED that all time references use this type. It is further  
1156 RECOMMENDED that all references be in UTC time. If, however, other time types are used,  
1157 then the *ValueType* attribute (described below) MUST be specified to indicate the data type of the  
1158 time format.

### 1159 **10.1 Model**

1160 This specification provides several tools for recipients to usprocess the expiration time presented  
1161 by the requestor. The first is the **creation time**. Recipients can use this value to assess possible  
1162 clock skew. However, to make some assessments, the time required to go from the requestor to  
1163 the recipient may also be useful in making this assessment. Two mechanisms are provided for  
1164 this. The first is that **intermediaries** may add timestamp elements indicating when they received  
1165 the message. This knowledge can be useful to get a holistic view of clocks along the message  
1166 path. The second is that intermediaries can specify any delays they imposed on message  
1167 delivery. It should be noted that not all **delays** can be accounted for, such as wire time and  
1168 parties that don't report. Recipients need to take this into account when evaluating clock skew.

### 1169 **10.2 Timestamp Elements**

1170 This specification defines the following message timestamp elements. These elements are  
1171 defined for use with the `<wsu:Timestamp>` header for SOAP messages, but they can be used  
1172 anywhere within the header or body that creation, expiration, and delay times are needed.

1173

#### 1174 **10.2.1 Creation**

1175 The `<wsu:Created>` element specifies a creation timestamp. The exact meaning and  
1176 semantics are dependent on the context in which the element is used. The syntax for this  
1177 element is as follows:

```
1178 <wsu:Created ValueType="..." wsu:Id="...">...</wsu:Created>
```

1179 The following describes the attributes and elements listed in the schema above:

1180 | `/wsu:Created` Deleted: /  
 1181 | This element's value is a creation timestamp. Its type is specified by the `ValueType`  
 1182 | attribute.

1183 | `/wsu:Created/@ValueType` Deleted: /  
 1184 | This optional attribute specifies the type of the time data. This is specified as the XML  
 1185 | Schema type. The default value is `xsd:dateTime`.

1186 | `/wsu:Created/@wsu:Id` Deleted: /  
 1187 | This optional attribute specifies an XML Schema ID that can be used to reference this  
 1188 | element.

## 1189 10.2.2 Expiration

1190 The `<wsu:Expires>` element specifies the expiration time. The exact meaning and processing  
 1191 rules for expiration depend on the context in which the element is used. The syntax for this  
 1192 element is as follows:

```
1193 <wsu:Expires ValueType="..." wsu:Id="...">...</wsu:Expires>
```

1194 The following describes the attributes and elements listed in the schema above:

1195 `/wsu:Expires`  
 1196 This element's value represents an expiration time. Its type is specified by the `ValueType`  
 1197 attribute

1198 | `/wsu:Expires/@ValueType` Deleted: /  
 1199 This optional attribute specifies the type of the time data. This is specified as the XML  
 1200 Schema type. The default value is `xsd:dateTime`.

1201 | `/wsu:Expires/@wsu:Id` Deleted: /  
 1202 This optional attribute specifies an XML Schema ID that can be used to reference this  
 1203 element.

1204 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,  
 1205 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's  
 1206 clock. The recipient, therefore, **MUST** make an assessment of the level of trust to be placed in  
 1207 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is  
 1208 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a  
 1209 judgment of the requestor's likely current clock time by means not described in this specification,  
 1210 for example an out-of-band clock synchronization protocol. The recipient may also use the  
 1211 creation time and the delays introduced by intermediate [SOAP](#) roles to estimate the degree of  
 1212 clock skew .

1213 One suggested formula for estimating clock skew is

```
1214 skew = recipient's arrival time - creation time - transmission time
```

1215 Transmission time may be estimated by summing the values of delay elements, if present. It  
 1216 should be noted that wire-time is only part of this if delays include it in estimates. Otherwise the  
 1217 transmission time will not reflect the on-wire time. If no delays are present, there are no special  
 1218 assumptions that need to be made about processing time

## 1219 10.3 Timestamp Header

1220 A `<wsu:Timestamp>` header provides a mechanism for expressing the creation and expiration  
 1221 times of a message introduced throughout the message path. Specifically, it uses the previously  
 1222 defined elements in the context of message creation, receipt, and processing.

1223 All times **SHOULD** be in UTC format as specified by the [XML Schema](#) type (`dateTime`). It should  
 1224 be noted that times support time precision as defined in the [XML Schema](#) specification.

1225 Multiple `<wsu:Timestamp>` headers can be specified if they are targeted at different SOAP  
 1226 roles. The ordering within the header is as illustrated below.  
 1227 The ordering of elements in this header is fixed and MUST be preserved by intermediaries.  
 1228 To preserve overall integrity of each `<wsu:Timestamp>` header, it is strongly RECOMMENDED  
 1229 that each SOAP role create or update the appropriate `<wsu:Timestamp>` header destined to  
 1230 itself.

1231 The schema outline for the `<wsu:Timestamp>` header is as follows:

```
1232 <wsu:Timestamp wsu:Id="...">
1233   <wsu:Created>...</wsu:Created>
1234   <wsu:Expires>...</wsu:Expires>
1235   ...
1236 </wsu:Timestamp>
```

1237 The following describes the attributes and elements listed in the schema above:

1238 `<wsu:Timestamp` Deleted: /

1239 This is the header for indicating message timestamps.

1240 `<wsu:Timestamp/Created` Deleted: /

1241 This represents the [creation time](#) of the message. This element is optional, but can only  
 1242 be specified once in a `Timestamp` header. Within the SOAP processing model, creation  
 1243 is the instant that the infoset is serialized for transmission. The creation time of the  
 1244 message SHOULD NOT differ substantially from its transmission time. The difference in  
 1245 time should be minimized.

1246 `<wsu:Timestamp/Expires` Deleted: /

1247 This represents the [expiration](#) of the message. This is optional, but can appear at most  
 1248 once in a `Timestamp` header. Upon expiration, the requestor asserts that the message  
 1249 is no longer valid. It is strongly RECOMMENDED that recipients (anyone who processes  
 1250 this message) discard (ignore) any message that has passed its expiration. A Fault code  
 1251 (`wsu:MessageExpired`) is provided if the recipient wants to inform the requestor that its  
 1252 message was expired. A service MAY issue a Fault indicating the message has expired.

1253 `<wsu:Timestamp/{any}` Deleted: /

1254 This is an extensibility mechanism to allow additional elements to be added to the  
 1255 header.

1256 `<wsu:Timestamp/@wsu:Id` Deleted: /

1257 This optional attribute specifies an XML Schema ID that can be used to reference this  
 1258 element.

1259 `<wsu:Timestamp/@{any}` Deleted: /

1260 This is an extensibility mechanism to allow additional attributes to be added to the  
 1261 header.

1262 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

```
1263 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1264   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
1265   <S:Header>
1266     <wsu:Timestamp>
1267       <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
1268       <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
1269     </wsu:Timestamp>
1270     ...
1271   </S:Header>
1272   <S:Body>
1273     ...
1274   </S:Body>
1275 </S:Envelope>
```

## 1276 10.4 TimestampTrace Header

1277 A `<wsu:TimestampTrace>` header provides a mechanism for expressing the delays introduced  
1278 throughout the message path. Specifically, it uses the previously defined elements in the context  
1279 of message creation, receipt, and processing.

1280 All times SHOULD be in UTC format as specified by the [XML Schema](#) type (`dateTime`). It should  
1281 be noted that times support time precision as defined in the [XML Schema](#) specification.

1282 Multiple `<wsu:TimestampTrace>` headers can be specified if they reference a different [SOAP](#)  
1283 role.

1284 The `<wsu:Received>` element specifies a receipt timestamp with an optional processing delay.

1285 The exact meaning and semantics are dependent on the context in which the element is used.

1286 It is also strongly RECOMMENDED that each [SOAP](#) role sign its elements by referencing their

1287 ID, NOT by signing the `TimestampTrace` header as the header is mutable.

1288 The syntax for this element is as follows:

```
1289 <wsu:TimestampTrace>  
1290   <wsu:Received Role="..." Delay="..." ValueType="..."  
1291     wsu:Id="..." >...</wsu:Received>  
1292 </wsu:TimestampTrace>
```

1293 The following describes the attributes and elements listed in the schema above:

1294 `/wsu:Received`

Deleted:

1295 This element's value is a receipt timestamp. The time specified SHOULD be a UTC  
1296 format as specified by the `ValueType` attribute (default is [XML Schema](#) type `dateTime`).

1297 `/wsu:Received/@Role`

Deleted:

1298 A required attribute, `Role`, indicates which [SOAP](#) role is indicating receipt. Roles MUST  
1299 include this attribute, with a value matching the role value as specified as a [SOAP](#)  
1300 intermediary.

1301 `/wsu:Received/@Delay`

Deleted:

1302 The value of this optional attribute is the delay associated with the [SOAP](#) role expressed  
1303 in milliseconds. The delay represents processing time by the `Role` after it received the  
1304 message, but before it forwarded to the next recipient.

1305 `/wsu:Received/@ValueType`

Deleted:

1306 This optional attribute specifies the type of the time data (the element value). This is  
1307 specified as the [XML Schema](#) type. If this attribute isn't specified, the default value is  
1308 `xsd:dateTime`.

1309 `/wsu:Received/@wsu:Id`

Deleted:

1310 This optional attribute specifies an [XML Schema](#) ID that can be used to reference this  
1311 element.

1312 The delay attribute indicates the time delay attributable to an [SOAP](#) role (intermediate  
1313 processor). In some cases this isn't known; for others it can be computed as *role's send time* –  
1314 *role's receipt time*.

1315 Each delay amount is indicated in units of milliseconds, without fractions. If a delay amount  
1316 would exceed the maximum value expressible in the datatype, the value should be set to the  
1317 maximum value of the datatype.

1318 The following example illustrates the use of the `<wsu:Timestamp>` header and a  
1319 `<wsu:TimestampTrace>` header indicating a processing delay of one minute subsequent to the  
1320 receipt which was two minutes after creation.

```
1321 <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"  
1322   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">  
1323   <S:Header>
```

```
1324 <wsu:Timestamp>
1325   <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
1326   <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
1327 </wsu:Timestamp>
1328 <wsu:TimespampTrace>
1329   <wsu:Received Role="http://x.com/" Delay="60000">
1330     2001-09-13T08:44:00Z</wsu:Received>
1331 </wsu:TimestampTrace>
1332 ...
1333 </S:Header>
1334 <S:Body>
1335 ...
1336 </S:Body>
1337 </S:Envelope>
1338
```

1339

## 11 Extended Example

1340

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

1341

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

1342

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

1343

<wsse:Security> header.

1344

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
1345      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
1346      xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
1347      xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility"
1348      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1349
(003)   <S:Header>
1350
(004)     <wsu:Timestamp>
1351
(005)       <wsu:Created wsu:Id="T0">
1352
(006)         2001-09-13T08:42:00Z
1353
(007)       </wsu:Created>
1354
(008)     </wsu:Timestamp>
1355
(009)     <wsse:Security>
1356
(010)       <wsse:BinarySecurityToken
1357         ValueType="wsse:X509v3"
1358         wsu:Id="X509Token"
1359         EncodingType="wsse:Base64Binary">
1360
(011)         MIIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1361
(012)       </wsse:BinarySecurityToken>
1362
(013)     <xenc:EncryptedKey>
1363
(014)       <xenc:EncryptionMethod Algorithm=
1364         "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
1365
(015)     <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
1366       ValueType="wsse:X509v3">MIGfMa0GCSq...
1367
(016)     </wsse:KeyIdentifier>
1368
(017)     <xenc:CipherData>
1369
(018)       <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1370
(019)     </xenc:CipherValue>
1371
(020)     </xenc:CipherData>
1372
(021)     <xenc:ReferenceList>
1373
(022)       <xenc:DataReference URI="#enc1"/>
1374
(023)     </xenc:ReferenceList>
1375
(024)     </xenc:EncryptedKey>
1376
(025)     <ds:Signature>
1377
(026)       <ds:SignedInfo>
1378
(027)         <ds:CanonicalizationMethod
1379           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1380
(028)       <ds:SignatureMethod
1381         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
1382
(029)       <ds:Reference URI="#T0">
1383
(030)       <ds:Transforms>
1384
(031)         <ds:Transform
1385           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1386
(032)       </ds:Transforms>
1387
(033)       <ds:DigestMethod
1388         Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
1389
(034)       <ds:DigestValue>LyLsF094hPi4wPU...
1390
(035)     </ds:DigestValue>
1391
(036)     </ds:Reference>
1392
(037)     <ds:Reference URI="#body">
1393
(038)     <ds:Transforms>
1394
(039)     <ds:Transform
1395
```

```

1396           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1397 (041)         </ds:Transforms>
1398 (042)         <ds:DigestMethod
1399           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1400 (043)         <ds:DigestValue>LyLsF094hPi4wPU...
1401 (044)         </ds:DigestValue>
1402 (045)         </ds:Reference>
1403 (046)       </ds:SignedInfo>
1404 (047)       <ds:SignatureValue>
1405 (048)         Hp1ZkmFZ/2kQLXDJbchm5gK...
1406 (049)       </ds:SignatureValue>
1407 (050)       <ds:KeyInfo>
1408 (051)         <wsse:SecurityTokenReference>
1409 (052)           <wsse:Reference URI="#X509Token" />
1410 (053)         </wsse:SecurityTokenReference>
1411 (054)       </ds:KeyInfo>
1412 (055)     </ds:Signature>
1413 (056)   </wsse:Security>
1414 (057) </S:Header>
1415 (058) <S:Body wsu:Id="body">
1416 (059)   <xenc:EncryptedData
1417     Type="http://www.w3.org/2001/04/xmlenc#Element"
1418     wsu:Id="enc1">
1419 (060)     <xenc:EncryptionMethod
1420     Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc" />
1421 (061)     <xenc:CipherData>
1422 (062)       <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1423 (063)     </xenc:CipherValue>
1424 (064)     </xenc:CipherData>
1425 (065)   </xenc:EncryptedData>
1426 (066) </S:Body>
1427 (067) </S:Envelope>

```

1428 Let's review some of the key sections of this example:

1429 Lines (003)-(057) contain the SOAP message headers.

1430 Lines (004)-(008) specify the timestamp information. In this case it indicates the creation time of  
1431 the message.

1432 Lines (009)-(056) represent the `<wsse:Security>` header block. This contains the security-  
1433 related information for the message.

1434 Lines (010)-(012) specify a [security token](#) that is associated with the message. In this case, it  
1435 specifies an [X.509](#) certificate that is encoded as Base64. Line (011) specifies the actual Base64  
1436 encoding of the certificate.

1437 Lines (013)-(025) specify the key that is used to encrypt the body of the message. Since this is a  
1438 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to  
1439 encrypt the key. Lines (015)-(017) specify the name of the key that was used to encrypt the  
1440 symmetric key. Lines (018)-(021) specify the actual encrypted form of the symmetric key. Lines  
1441 (022)-(024) identify the encryption block in the message that uses this symmetric key. In this  
1442 case it is only used to encrypt the body (Id="enc1").

1443 Lines (026)-(055) specify the digital signature. In this example, the signature is based on the  
1444 [X.509](#) certificate. Lines (027)-(046) indicate what is being signed. Specifically, Line (039)  
1445 references the creation timestamp and line (038) references the message body.

1446 Lines (047)-(049) indicate the actual signature value – specified in Line (042).

1447 Lines (051)-(053) indicate the key that was used for the signature. In this case, it is the [X.509](#)  
1448 certificate included in the message. Line (052) provides a URI link to the Lines (010)-(012).

1449 The body of the message is represented by Lines (056)-(066).

1450 Lines (059)-(065) represent the encrypted metadata and form of the body using [XML Encryption](#).  
1451 Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line

1452 (060) specifies the encryption algorithm – Triple-DES in this case. Lines (062)-(063) contain the  
1453 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the  
1454 key as the key references this encryption – Line (023).



## 1455 12Error Handling

1456 There are many circumstances where an *error* can occur while processing security information.  
1457 For example:

- 1458 • Invalid or unsupported type of security token, signing, or encryption
- 1459 • Invalid or unauthenticated or unauthenticatable security token
- 1460 • Invalid signature
- 1461 • Decryption failure
- 1462 • Referenced security token is unavailable
- 1463 • Unsupported namespace

1464 These can be grouped into two *classes* of errors: unsupported and failure. For the case of  
1465 unsupported errors, the recipient *MAY* provide a response that informs the sender of supported  
1466 formats, etc. For failure errors, the recipient *MAY* choose not to respond, as this may be a form  
1467 of Denial of Service (DOS) or cryptographic attack. We combine signature and encryption  
1468 failures to mitigate certain types of attacks.

1469 If a failure is returned to a sender then the failure *MUST* be reported using *SOAPs* Fault  
1470 mechanism. The following tables outline the predefined security fault codes. The "unsupported"  
1471 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

1472 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

1473

1474 It is strongly RECOMMENDED that messages include digitally signed elements to allow message  
1475 recipients to detect replays of the message when the messages are exchanged via an open  
1476 network. These can be part of the message or of the headers defined from other SOAP  
1477 extensions. Four typical approaches are:

- 1478 • Timestamp
- 1479 • Sequence Number
- 1480 • Expirations
- 1481 • Message Correlation

1482 This specification defines the use of XML Signature and XML Encryption in SOAP headers. As  
1483 one of the building blocks for securing SOAP messages, it is intended to be used in conjunction  
1484 with other security techniques. Digital signatures need to be understood in the context of other  
1485 security mechanisms and possible threats to an entity.

1486 Digital signatures alone do not provide message authentication. One can record a signed  
1487 message and resend it (a replay attack). To prevent this type of attack, digital signatures must be  
1488 combined with an appropriate means to ensure the uniqueness of the message, such as  
1489 timestamps or sequence numbers (see earlier section for additional details).

1490 When digital signatures are used for verifying the identity of the sending party, the sender must  
1491 prove the possession of the private key. One way to achieve this is to use a challenge-response  
1492 type of protocol. Such a protocol is outside the scope of this document.

1493 To this end, the developers can attach timestamps, expirations, and sequences to messages.

1494 Implementers should also be aware of all the security implications resulting from the use of digital  
1495 signatures in general and XML Signature in particular. When building trust into an application  
1496 based on a digital signature there are other technologies, such as certificate evaluation, that must  
1497 be incorporated, but these are outside the scope of this document.

1498 Requestors should use digital signatures to sign security tokens that do not include signatures (or  
1499 other protection mechanisms) to ensure that they have not been altered in transit. It is strongly  
1500 RECOMMENDED that all relevant and immutable message content be signed by the sender.  
1501 Receivers SHOULD only consider those portions of the document that are covered by the  
1502 sender's signature as being subject to the assertions in the message. Security tokens appearing  
1503 in <wsse:Security> header elements SHOULD be signed by their issuing authority so that  
1504 message receivers can have confidence that the assertions have not been forged or altered since  
1505 their issuance. It is strongly RECOMMENDED that a message sender sign any  
1506 <SecurityToken> elements that it is confirming and that are not signed by their issuing  
1507 authority.

Formatted: Code Embedded, ce,  
Font: (Default) Arial

Formatted: Font: (Default) Arial

Formatted: Code Embedded, ce,  
Font: (Default) Arial

Formatted: Font: (Default) Arial

1508 Also, as described in XML Encryption, we note that the combination of signing and encryption  
1509 over a common data item may introduce some cryptographic vulnerability. For example,  
1510 encrypting digitally signed data, while leaving the digital signature in the clear, may allow plain  
1511 text guessing attacks. The proper usage of nonce guards against replay attacks.

1512 In order to trust IDs and timestamps, they SHOULD be signed using the mechanisms outlined in  
1513 this specification. This allows readers of the IDs and timestamps information to be certain that  
1514 the IDs and timestamps haven't been forged or altered in any way. It is strongly  
1515 RECOMMENDED that IDs and timestamp elements be signed.

1516 Timestamps can also be used to mitigate replay attacks. Signed timestamps MAY be used to  
1517 keep track of messages (possibly by caching the most recent timestamp from a specific service)  
1518 and detect replays of previous messages. It is RECOMMENDED that timestamps and nonces be  
1519 cached for a given period of time, as a guideline a value of five minutes can be used as a

1520 minimum to detect replays, and that timestamps older than that given period of time set be  
1521 rejected. in interactive scenarios.

1522 When a password in a <UsernameToken> is used for authentication, the password needs to be  
1523 properly protected. If the underlying transport does not provide enough protection against  
1524 eavesdropping, the password SHOULD be digested as described in Section 6.1.1. Even so, the  
1525 password must be strong enough so that simple password guessing attacks will not reveal the  
1526 secret from a captured message.

1527 In one-way message authentication, it is RECOMMENDED that the sender and the recipient re-  
1528 use the elements and structure defined in this specification for proving and validating freshness of  
1529 a message. It is RECOMMEND that the nonce value be unique per message (never been used  
1530 as a nonce before by the sender and recipient) and use the <wsse:Nonce> element within the  
1531 <wsse:Security> header. Further, the <wsu:Timestamp> header SHOULD be used with a  
1532 <wsu:Created> element. It is strongly RECOMMENDED that the <wsu:Created> ,  
1533 <wsse:Nonce> elements be included in the signature.

Deleted: ¶

Deleted:

Deleted: .

1534 **14 Privacy Considerations**

1535 TBD

## 1536 **15 Acknowledgements**

1537 This specification was developed as a result of joint work of many individuals from the WSS TC  
1538 including: TBD

1539 The input specifications for this document were developed as a result of joint work with many  
1540 individuals and teams, including: Keith Ballinger, Microsoft, Bob Blakley, IBM, Allen Brown,  
1541 Microsoft, Joel Farrell, IBM, Mark Hayes, VeriSign, Kelvin Lawrence, IBM, Scott Konersmann,  
1542 Microsoft, David Melgar, IBM, Dan Simon, Microsoft, Wayne Vicknair, IBM.

1543

## 16References

- 1544       **[DIGSIG]**           Informational RFC 2828, "[Internet Security Glossary](#)," May 2000.
- 1545       **[Kerberos]**           J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," [RFC 1510](#), September 1993, <http://www.ietf.org/rfc/rfc1510.txt>.
- 1546
- 1547       **[KEYWORDS]**        S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," [RFC 2119](#), Harvard University, March 1997
- 1548
- 1549       **[SHA -1]**            FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce / National Institute of Standards and Technology. <http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt>
- 1550
- 1551
- 1552       **[SOAP11]**            W3C Note, "[SOAP: Simple Object Access Protocol 1.1](#)," 08 May 2000.
- 1553       **[SOAP12]**            **W3C Working Draft**, "SOAP Version 1.2 Part 1: Messaging Framework", 26 June 2002
- 1554
- 1555       **[SOAP-SEC]**        W3C Note, "[SOAP Security Extensions: Digital Signature](#)," 06 February 2001.
- 1556
- 1557       **[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.
- 1558
- 1559
- 1560       **[WS-Security]**       "[Web Services Security Language](#)", IBM, Microsoft, VeriSign, April 2002.
- 1561       "[WS-Security Addendum](#)", IBM, Microsoft, VeriSign, August 2002.
- 1562       "[WS-Security XML Tokens](#)", IBM, Microsoft, VeriSign, August 2002.
- 1563       **[XML-C14N]**        W3C Recommendation, "[Canonical XML Version 1.0](#)," 15 March 2001
- 1564       **[XML-Encrypt]**     W3C Working Draft, "[XML Encryption Syntax and Processing](#)," 04 March 2002.
- 1565
- 1566       **[XML-ns]**            W3C Recommendation, "[Namespaces in XML](#)," 14 January 1999.
- 1567       **[XML-Schema]**     W3C Recommendation, "[XML Schema Part 1: Structures](#)," 2 May 2001.
- 1568       W3C Recommendation, "[XML Schema Part 2: Datatypes](#)," 2 May 2001.
- 1569       **[XML Signature]** W3C Recommendation, "[XML Signature Syntax and Processing](#)," 12 February 2002.
- 1570
- 1571       **[X509]**            S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified Certificates Profile,"
- 1572       <http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I>
- 1573
- 1574
- 1575       **[XPath]**            W3C Recommendation, "[XML Path Language](#)", 16 November 1999
- 1576       **[WSS-SAML]**        OASIS Working Draft 02, "Web Services Security SAML Token Binding, 23 September 2002
- 1577
- 1578       **[WSS-XrML]**        OASIS Working Draft 01, "Web Services Security XrML Token Binding, 20 September 2002
- 1579
- 1580       **[WSS-X509]**        OASIS Working Draft 01, "Web Services Security X509 Binding, 18 September 2002
- 1581
- 1582       **[WSS-Kerberos]**   OASIS Working Draft 01, "Web Services Security Kerberos Binding, 18 September 2002
- 1583

1584       **[XPointer]** "XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation",  
1585 DeRose, Maler, Daniel, 11 September 2001.  
1586  
1587

Formatted: Bullets and Numbering

1588

## **Appendix A: Utility Elements and Attributes**

1589  
1590  
1591  
1592  
1593

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

Formatted: Bullets and Numbering

1594

### **A.1. Identification Attribute**

1595  
1596  
1597  
1598  
1599  
1600

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

1601  
1602  
1603  
1604

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

1605  
1606  
1607

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

1608

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

Formatted: Bullets and Numbering

1609

### **A.2. Timestamp Elements**

1610  
1611  
1612

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

1613  
1614  
1615  
1616  
1617

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

1618

The following table provides an overview of these elements:

<b><u>Element</u></b>	<b><u>Description</u></b>
<u>&lt;wsu:Created&gt;</u>	This element is used to indicate the creation time associated with the enclosing context.
<u>&lt;wsu:Expires&gt;</u>	This element is used to indicate the expiration time associated with the enclosing context.
<u>&lt;wsu:Received&gt;</u>	This element is used to indicate the receipt time reference associated with the enclosing context.

1619

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



1620  
1621  
1622  
1623  
1624  
1625

### **A.3. General Schema Types**

The schema for the utility aspects of this specification also defines some general purpose schema elements. While these elements are used in the schema for the specification, they are general purpose and can be used by other specifications to have common time types.

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

<b><u>Schema Element</u></b>	<b><u>Description</u></b>
<u>wsu:commonAtts attribute group</u>	<u>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.</u>
<u>wsu:AttributedDateTime type</u>	<u>This type extends the XML Schema dateTime type to include the common attributes.</u>
<u>wsu:AttributedURI type</u>	<u>This type extends the XML Schema dateTime type to include the common attributes.</u>

Formatted: Bullets and Numbering

Formatted: Normal

Formatted: AppendixHeading1

1626

## Appendix B: SecurityTokenReference Model

1627

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

1628

1629

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

1630

1631

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

Formatted: Bullets and Numbering

1632

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

1633

1634

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

1635

1636

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

1637

1638

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

1639

1640

1641

The following use cases drive the above motivations:

1642

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

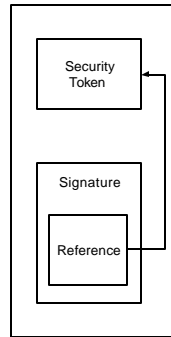
Formatted: Definition term Char, Check spelling and grammar

1643

1644

1645

Formatted: Bullets and Numbering



1646

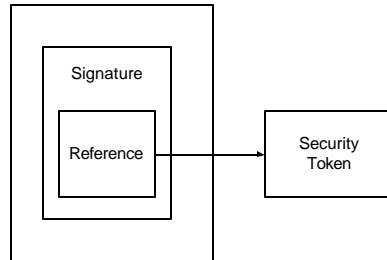
- **Remote Reference** – A security token, that is not included in the message but may be available at a specific URI, is associated with an XML Signature. The figure below illustrates this:

Formatted: Definition term Char, No underline

1647

1648

1649

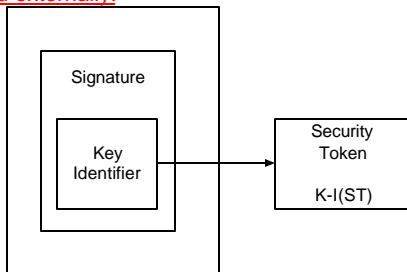


1650

1651  
1652  
1653  
1654

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

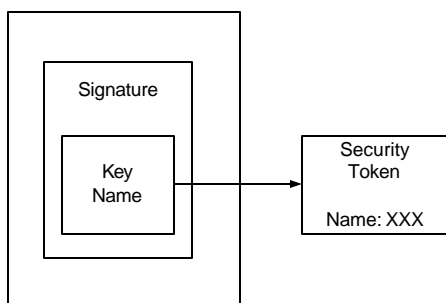
Formatted: Definition term Char, No underline



1655  
1656  
1657  
1658  
1659

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

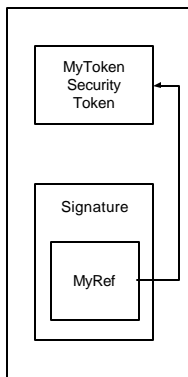
Formatted: Definition term Char, No underline



1660  
1661  
1662  
1663

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

Formatted: Definition term Char, No underline

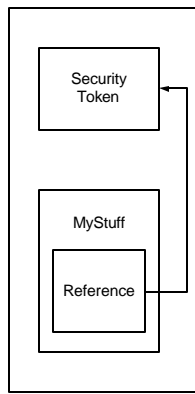


1664  
1665  
1666  
1667  
1668  
1669

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

Formatted: Definition term Char, No underline

Formatted: Bullets and Numbering



1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708

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

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

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

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

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

Formatted: Definition term Char, No underline

Formatted: Bullets and Numbering

Formatted: Definition term Char, No underline

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

The following provides additional details on the specific reference mechanisms defined in WS-Security:

- **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to the security token. If only the fragment is specified, then it references the security token within the document whose *wsu:Id* matches the fragment. For non-fragment URIs, the reference is to a [potentially external] security token identified using a URI. There are no implied semantics around the processing of the URI.
- **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token by specifying a known value (identifier) for the token, which is determined by applying a special function to the security token (e.g. a hash of key fields). This approach is typically unique for the specific security token but requires a profile or token-specific function to be specified. The *ValueType* attribute provide a hint as to the desired token type. The *EncodingType* attribute specifies how the unique value (identifier) is encoded. For example, a hash value may be encoded using base 64 encoding (the default).

Formatted: Definition term Char, No underline

Formatted: Bullets and Numbering

Formatted: Definition term Char, No underline

1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721

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

Formatted: Definition term Char, No underline

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

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

Formatted: Bullets and Numbering

Formatted: Normal

1722

## **Appendix C: Revision History**

<b>Rev</b>	<b>Date</b>	<b>What</b>
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

1723

1724

## **Appendix D: Notices**

1725 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
1726 that might be claimed to pertain to the implementation or use of the technology described in this  
1727 document or the extent to which any license under such rights might or might not be available;  
1728 neither does it represent that it has made any effort to identify any such rights. Information on  
1729 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
1730 website. Copies of claims of rights made available for publication and any assurances of licenses  
1731 to be made available, or the result of an attempt made to obtain a general license or permission  
1732 for the use of such proprietary rights by implementors or users of this specification, can be  
1733 obtained from the OASIS Executive Director.

1734 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
1735 applications, or other proprietary rights which may cover technology that may be required to  
1736 implement this specification. Please address the information to the OASIS Executive Director.

1737 Copyright © OASIS Open 2002. *All Rights Reserved.*

1738 This document and translations of it may be copied and furnished to others, and derivative works  
1739 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
1740 published and distributed, in whole or in part, without restriction of any kind, provided that the  
1741 above copyright notice and this paragraph are included on all such copies and derivative works.  
1742 However, this document itself does not be modified in any way, such as by removing the  
1743 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS  
1744 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual  
1745 Property Rights document must be followed, or as required to translate it into languages other  
1746 than English.

1747 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
1748 successors or assigns.

1749 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
1750 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
1751 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE  
1752 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
1753 PARTICULAR PURPOSE.

1754