1

Web Services Security:
SOAP Message Security 1.0
(WS-Security 2004)

OASIS Standard 200401, March 2004

Document identifier:
{*WSS: SOAP Message Security* }-{*1.0*} (Word) (PDF)

Document Location:
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0

Errata Location:
http://www.oasis-open.org/committees/wss

Editors:

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

Contributors:

Gene	Thurston	AmberPoint
Frank	Siebenlist	Argonne National Lab
Merlin	Hughes	Baltimore Technologies
Irving	Reid	Baltimore Technologies
Peter	Dapkus	BEA
Hal	Lockhart	BEA
Symon	Chang	CommerceOne
Srinivas	Davanum	Computer Associates
Thomas	DeMartini	ContentGuard
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard
Shawn	Sharp	Cyclone Commerce
Ganesh	Vaideeswaran	Documentum
Sam	Wei	Documentum
John	Hughes	Entegrity
Tim	Moses	Entrust
Toshihiro	Nishimura	Fujitsu
Tom	Rutt	Fujitsu
Yutaka	Kudo	Hitachi
Jason	Rouault	HP
Paula	Austel	IBM
Bob	Blakley	IBM

Joel	Farrell	IBM
Satoshi	Hada	IBM
Maryann	Hondo	IBM
Michael	McIntosh	IBM
Hiroshi	Maruyama	IBM
David	Melgar	IBM
Anthony	Nadalin	IBM
Nataraj	Nagaratnam	IBM
Wayne	Vicknair	IBM
Kelvin	Lawrence	IBM (co-Chair)
Don	Flinn	Individual
Bob	Morgan	Individual
Bob	Atkinson	Microsoft
Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Paul	Cotton	Microsoft
Giovanni	Della-Libera	Microsoft
Vijay	Gajjala	Microsoft
Johannes	Klein	Microsoft
Scott	Konersmann	Microsoft
Chris	Kurt	Microsoft
Brian	LaMacchia	Microsoft
Paul	Leach	Microsoft
John	Manferdelli	Microsoft
John	Shewchuk	Microsoft
Dan	Simon	Microsoft
Hervey	Wilson	Microsoft
Chris	Kaler	Microsoft (co-Chair)
Prateek	Mishra	Netegrity
Frederick	Hirsch	Nokia
Senthil	Sengodan	Nokia
Lloyd	Burch	Novell
Ed	Reed	Novell
Charles	Knouse	Oblix
Steve	Anderson	OpenNetwork (Sec)
Vipin	Samar	Oracle
Jerry	Schwarz	Oracle
Eric	Gravengaard	Reactivity
Stuart	King	Reed Elsevier
Andrew	Nash	RSA Security
Rob	Philpott	RSA Security
Peter	Rostin	RSA Security
Martijn	de Boer	SAP
Blake	Dournaee	Sarvega
Pete	Wenzel	SeeBeyond
Jonathan	Tourzan	Sony
Yassir	Elley	Sun Microsystems
Jeff	Hodges	Sun Microsystems
Ronald	Monzillo	Sun Microsystems
Jan	Alexander	Systinet
Michael	Nguyen	The IDA of Singapore
Don	Adams	TIBCO

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

14

Abstract:

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

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

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

28 **Status:**

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

Table of Contents

112 1 Introduction

113 This OASIS specification is the result of significant new work by the WSS Technical Committee
114 and supersedes the input submissions, Web Service Security (WS-Security) Version 1.0 April 5,
115 2002 and Web Services Security Addendum Version 1.0 August 18, 2002.
116 This specification proposes a standard set of SOAP [SOAP11, SOAP12] extensions that can be
117 used when building secure Web services to implement message content integrity and
118 confidentiality. This specification refers to this set of extensions and modules as the "Web
119 Services Security: SOAP Message Security" or "WSS: SOAP Message Security".
120 This specification is flexible and is designed to be used as the basis for securing Web services
121 within a wide variety of security models including PKI, Kerberos, and SSL. Specifically, this
122 specification provides support for multiple security token formats, multiple trust domains, multiple
123 signature formats, and multiple encryption technologies. The token formats and semantics for
124 using these are defined in the associated profile documents.
125 This specification provides three main mechanisms: ability to send security tokens as part of a
126 message, message integrity, and message confidentiality. These mechanisms by themselves do
127 not provide a complete security solution for Web services. Instead, this specification is a building
128 block that can be used in conjunction with other Web service extensions and higher-level
129 application-specific protocols to accommodate a wide variety of security models and security
130 technologies.
131 These mechanisms can be used independently (e.g., to pass a security token) or in a tightly
132 coupled manner (e.g., signing and encrypting a message or part of a message and providing a
133 security token or token path associated with the keys used for signing and encryption).

134 1.1 Goals and Requirements

135 The goal of this specification is to enable applications to conduct secure SOAP message
136 exchanges.
137 This specification is intended to provide a flexible set of mechanisms that can be used to
138 construct a range of security protocols; in other words this specification intentionally does not
139 describe explicit fixed security protocols.
140 As with every security protocol, significant efforts must be applied to ensure that security
141 protocols constructed using this specification are not vulnerable to any one of a wide range of
142 attacks. The examples in this specification are meant to illustrate the syntax of these mechanisms
143 and are not intended as examples of combining these mechanisms in secure ways.
144 The focus of this specification is to describe a single-message security language that provides for
145 message security that may assume an established session, security context and/or policy
146 agreement.
147 The requirements to support secure message exchange are listed below.

148 1.1.1 Requirements

149 The Web services security language must support a wide variety of security models. The
150 following list identifies the key driving requirements for this specification:
151   • Multiple security token formats
152   • Multiple trust domains
153   • Multiple signature formats
154   • Multiple encryption technologies
155   • End-to-end message content security and not just transport-level security

156 1.1.2 Non-Goals

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

158   • Establishing a security context or authentication mechanisms.
159   • Key derivation.
160   • Advertisement and exchange of security policy.
161   • How trust is established or determined.
162   • Non-repudiation.
163

164 2 Notations and Terminology

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

166   2.1 Notational Conventions

167   The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
168   "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be
169   interpreted as described in RFC 2119.
170   When describing abstract data models, this specification uses the notational convention used by
171   the XML Infoset. Specifically, abstract property names always appear in square brackets (e.g.,
172   [some property]).
173   When describing concrete XML schemas, this specification uses a convention where each
174   member of an element's [children] or [attributes] property is described using an XPath-like
175   notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence
176   of an element wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute
177   wildcard (<xs:anyAttribute/>).
178   Readers are presumed to be familiar with the terms in the Internet Security Glossary [GLOS].

179   2.2 Namespaces

180   Namespace URIs (of the general form "some-URI") represents some application-dependent or
181   context-dependent URI as defined in RFC 2396 [URI]. The XML namespace URIs that MUST be
182   used by implementations of this specification are as follows (note that elements used in this
183   specification are from various namespaces):
184
```
185           http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
186   wssecurity-secext-1.0.xsd
187           http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
188   wssecurity-utility-1.0.xsd
```
189
190   This specification is designed to work with the general SOAP [SOAP11, SOAP12] message
191   structure and message processing model, and should be applicable to any version of SOAP. The
192   current SOAP 1.1 namespace URI is used herein to provide detailed examples, but there is no
193   intention to limit the applicability of this specification to a single version of SOAP.
194   The namespaces used in this document are shown in the following table (note that for brevity, the
195   examples use the prefixes listed below but do not include the URIs – those listed below are
196   assumed).
197

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

xenc	http://www.w3.org/2001/04/xmlenc#

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

199 **2.3 Acronyms and Abbreviations**

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

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

201 **2.4 Terminology**

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

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

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

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

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

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

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

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

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

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

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

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

231

Security Tokens	
Unsigned Security Tokens	**Signed Security Tokens**
→ Username	→ X.509 Certificates → Kerberos tickets
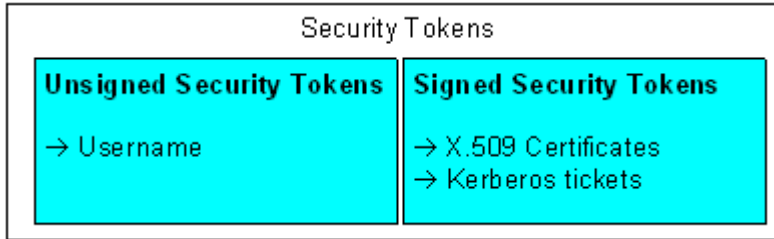
232
233
234 **Signed Security Token** – A *signed security token* is a security token that is asserted and
235 cryptographically signed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket).
236 **Trust -** *Trust is* the characteristic that one entity is willing to rely upon a second entity to execute
237 a set of actions and/or to make set of assertions about a set of subjects and/or scopes.
238
239
240

3 Message Protection Mechanisms

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

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

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

3.1 Message Security Model

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

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

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

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

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

3.2 Message Protection

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

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

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

This document defines syntax and semantics of signatures within a `<wsse:Security>` element. This document does not specify any signature appearing outside of a `<wsse:Security>` element.

3.3 Invalid or Missing Claims

A message recipient SHOULD reject messages containing invalid signatures, messages missing necessary claims or messages whose claims have unacceptable values. Such messages are unauthorized (or malformed). This specification provides a flexible way for the message producer

287 to make a claim about the security properties by associating zero or more security tokens with the
288 message. An example of a security claim is the identity of the producer; the producer can claim
289 that he is Bob, known as an employee of some company, and therefore he has the right to send
290 the message.

3.4 Example

292 The following example illustrates the use of a custom security token and associated signature.
293 The token contains base64 encoded binary data conveying a symmetric key which, we assume,
294 can be properly authenticated by the recipient. The message producer uses the symmetric key
295 with an HMAC signing algorithm to sign the message. The message receiver uses its knowledge
296 of the shared secret to repeat the HMAC key calculation which it uses to validate the signature
297 and in the process confirm that the message was authored by the claimed user identity.
298

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
            xmlns:ds="...">
(003)   <S11:Header>
(004)       <wsse:Security
              xmlns:wsse="...">
(005)          <xxx:CustomToken wsu:Id="MyID"
                             xmlns:xxx="http://fabrikam123/token">
(006)              FHUIORv...
(007)          </xxx:CustomToken>
(008)          <ds:Signature>
(009)             <ds:SignedInfo>
(010)                 <ds:CanonicalizationMethod
                         Algorithm=
                           "http://www.w3.org/2001/10/xml-exc-c14n#"/>
(011)                 <ds:SignatureMethod
                         Algorithm=
                           "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
(012)                 <ds:Reference URI="#MsgBody">
(013)                    <ds:DigestMethod
                            Algorithm=
                              "http://www.w3.org/2000/09/xmldsig#sha1"/>
(014)                    <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
(015)                 </ds:Reference>
(016)             </ds:SignedInfo>
(017)             <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
(018)             <ds:KeyInfo>
(019)                 <wsse:SecurityTokenReference>
(020)                    <wsse:Reference URI="#MyID"/>
(021)                 </wsse:SecurityTokenReference>
(022)             </ds:KeyInfo>
(023)          </ds:Signature>
(024)       </wsse:Security>
(025)   </S11:Header>
(026)   <S11:Body wsu:Id="MsgBody">
(027)     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
              QQQ
            </tru:StockSymbol>
(028)   </S11:Body>
(029) </S11:Envelope>
```

339
340 The first two lines start the SOAP envelope. Line (003) begins the headers that are associated
341 with this SOAP message.
342 Line (004) starts the `<wsse:Security>` header defined in this specification. This header
343 contains security information for an intended recipient. This element continues until line (024).

344 Lines (005) to (007) specify a custom token that is associated with the message. In this case, it
345 uses an externally defined custom token format.
346 Lines (008) to (023) specify a digital signature. This signature ensures the integrity of the signed
347 elements. The signature uses the XML Signature specification identified by the ds namespace
348 declaration in Line (002).
349 Lines (009) to (016) describe what is being signed and the type of canonicalization being used.
350 Line (010) specifies how to canonicalize (normalize) the data that is being signed. Lines (012) to
351 (015) select the elements that are signed and how to digest them. Specifically, line (012)
352 indicates that the `<S11:Body>` element is signed. In this example only the message body is
353 signed; typically all critical elements of the message are included in the signature (see the
354 Extended Example below).
355 Line (017) specifies the signature value of the canonicalized form of the data that is being signed
356 as defined in the XML Signature specification.
357 Lines (018) to (022) provides information, partial or complete, as to where to find the security
358 token associated with this signature. Specifically, lines (019) to (021) indicate that the security
359 token can be found at (pulled from) the specified URL.
360 Lines (026) to (028) contain the body (payload) of the SOAP message.
361

362 4 ID References

363   There are many motivations for referencing other message elements such as signature
364   references or correlating signatures to security tokens. For this reason, this specification defines
365   the `wsu:Id` attribute so that recipients need not understand the full schema of the message for
366   processing of the security elements. That is, they need only "know" that the `wsu:Id` attribute
367   represents a schema type of ID which is used to reference elements. However, because some
368   key schemas used by this specification don't allow attribute extensibility (namely XML Signature
369   and XML Encryption), this specification also allows use of their local ID attributes in addition to
370   the `wsu:Id` attribute. As a consequence, when trying to locate an element referenced in a
371   signature, the following attributes are considered:
372   • Local ID attributes on XML Signature elements
373   • Local ID attributes on XML Encryption elements
374   • Global wsu:Id attributes (described below) on elements
375   In addition, when signing a part of an envelope such as the body, it is RECOMMENDED that an
376   ID reference is used instead of a more general transformation, especially XPath [XPATH]. This is
377   to simplify processing.

378 4.1 Id Attribute

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

393 4.2 Id Schema

394   To simplify the processing for intermediaries and recipients, a common attribute is defined for
395   identifying an element. This attribute utilizes the XML Schema ID type and specifies a common
396   attribute for indicating this information for elements.
397   The syntax for this attribute is as follows:
398
399   `<anyElement wsu:Id="...">...</anyElement>`
400
401   The following describes the attribute illustrated above:
402   *.../@wsu:Id*
403   This attribute, defined as type `xsd:ID`, provides a well-known attribute for specifying the
404   local ID of an element.
405   Two `wsu:Id` attributes within an XML document MUST NOT have the same value.
406   Implementations MAY rely on XML Schema validation to provide rudimentary enforcement for
407   intra-document uniqueness. However, applications SHOULD NOT rely on schema validation
408   alone to enforce uniqueness.

409 This specification does not specify how this attribute will be used and it is expected that other
410 specifications MAY add additional semantics (or restrictions) for their usage of this attribute.
411 The following example illustrates use of this attribute to identify an element:
412

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

415
416 Conformant processors that do support XML Schema MUST treat this attribute as if it was
417 defined using a global attribute declaration.
418 Conformant processors that do not support dynamic XML Schema or DTDs discovery and
419 processing are strongly encouraged to integrate this attribute definition into their parsers. That is,
420 to treat this attribute information item as if its PSVI has a [type definition] which {target
421 namespace} is "`http://www.w3.org/2001/XMLSchema`" and which {name} is "Id." Doing so
422 allows the processor to inherently know *how* to process the attribute without having to locate and
423 process the associated schema. Specifically, implementations MAY support the value of the
424 `wsu:Id` as the valid identifier for use as an XPointer [XPointer] shorthand pointer for
425 interoperability with XML Signature references.

426 5 Security Header

427 The `<wsse:Security>` header block provides a mechanism for attaching security-related
428 information targeted at a specific recipient in the form of a SOAPactor/role. This may be either
429 the ultimate recipient of the message or an intermediary. Consequently, elements of this type
430 may be present multiple times in a SOAP message. An active intermediary on the message path
431 MAY add one or more new sub-elements to an existing `<wsse:Security>` header block if they
432 are targeted for its SOAP node or it MAY add one or more new headers for additional targets.
433 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted
434 for separate recipients. However, only one `<wsse:Security>` header block MAY omit the S11:
435 actor or `S12:role` attributes. Two `<wsse:Security>` header blocks MUST NOT have the
436 same value for S11:actor or `S12:role`. Message security information targeted for different
437 recipients MUST appear in different `<wsse:Security>` header blocks. This is due to potential
438 processing order issues (e.g. due to possible header re-ordering). The `<wsse:Security>`
439 header block without a specified S11:actor or `S12:role` MAY be processed by anyone, but
440 MUST NOT be removed prior to the final destination or endpoint.
441 As elements are added to a `<wsse:Security>` header block, they SHOULD be prepended to
442 the existing elements. As such, the `<wsse:Security>` header block represents the signing and
443 encryption steps the message producer took to create the message. This prepending rule
444 ensures that the receiving application can process sub-elements in the order they appear in the
445 `<wsse:Security>` header block, because there will be no forward dependency among the sub-
446 elements. Note that this specification does not impose any specific order of processing the sub-
447 elements. The receiving application can use whatever order is required.
448 When a sub-element refers to a key carried in another sub-element (for example, a signature
449 sub-element that refers to a binary security token sub-element that contains the X.509 certificate
450 used for the signature), the key-bearing element SHOULD be ordered to precede the key-using
451 Element:
452
```
453     <S11:Envelope>
454         <S11:Header>
455             ...
456             <wsse:Security S11:actor="..." S11:mustUnderstand="...">
457                 ...
458             </wsse:Security>
459             ...
460         </S11:Header>
461         ...
462     </S11:Envelope>
```
463
464 The following describes the attributes and elements listed in the example above:
465 */wsse:Security*
466   This is the header block for passing security-related message information to a recipient.
467 */wsse:Security/@S11:actor*
468   This attribute allows a specific SOAP 1.1 [SPOAP11] actor to be identified. This attribute
469   is optional; however, no two instances of the header block may omit a actor or specify the
470   same actor.
471 */wsse:Security/@S12:role*
472   This attribute allows a specific SOAP 1.2 [SOAP12] role to be identified. This attribute is
473   optional; however, no two instances of the header block may omit a role or specify the
474   same role.
475
476 */wsse:Security/{any}*

477   This is an extensibility mechanism to allow different (extensible) types of security
478   information, based on a schema, to be passed. Unrecognized elements SHOULD cause
479   a fault.
480 */wsse:Security/@{any}*
481   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
482   added to the header. Unrecognized attributes SHOULD cause a fault.
483 All compliant implementations MUST be able to process a `<wsse:Security>` element.
484 All compliant implementations MUST declare which profiles they support and MUST be able to
485 process a `<wsse:Security>` element including any sub-elements which may be defined by that
486 profile. It is RECOMMENDED that undefined elements within the `<wsse:Security>` header
487 not be processed.
488 The next few sections outline elements that are expected to be used within a `<wsse:Security>`
489 header.
490 When a `<wsse:Security>` header includes a `mustUnderstand="true"` attribute:
491   •   The receiver MUST generate a SOAP fault if does not implement the WSS: SOAP
492   Message Security specification corresponding to the namespace. Implementation means
493   ability to interpret the schema as well as follow the required processing rules specified in
494   WSS: SOAP Message Security.
495   •   The receiver must generate a fault if unable to interpret or process security tokens
496   contained in the `<wsse:Security>` header block according to the corresponding WSS:
497   SOAP Message Security token profiles.
498   •   Receivers MAY ignore elements or extensions within the `<wsse:Security>` element,
499   based on local security policy.

500 6 Security Tokens

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

503 6.1 Attaching Security Tokens

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

509 6.1.1 Processing Rules

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

514 6.1.2 Subject Confirmation

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

518 6.2 User Name Token

519 6.2.1 Usernames

520 The `<wsse:UsernameToken>` element is introduced as a way of providing a username. This
521 element is optionally included in the `<wsse:Security>` header.
522 The following illustrates the syntax of this element:
523
```
524     <wsse:UsernameToken wsu:Id="...">
525         <wsse:Username>...</wsse:Username>
526     </wsse:UsernameToken>
```
527
528 The following describes the attributes and elements listed in the example above:
529 */wsse:UsernameToken*
530   This element is used to represent a claimed identity.
531 */wsse:UsernameToken/@wsu:Id*
532   A string label for this security token.
533 */wsse:UsernameToken/wsse:Username*
534   This required element specifies the claimed identity.
535 */wsse:UsernameToken/wsse:Username/@{any}*
536   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
537   added to the `<wsse:Username>` element.
538 /wsse:UsernameToken/{any}
539   This is an extensibility mechanism to allow different (extensible) types of security
540   information, based on a schema, to be passed. Unrecognized elements SHOULD cause
541   a fault.
542 */wsse:UsernameToken/@{any}*

543   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
544   added to the `<wsse:UsernameToken>` element. Unrecognized attributes SHOULD
545   cause a fault.
546 All compliant implementations MUST be able to process a `<wsse:UsernameToken>` element.
547 The following illustrates the use of this:
548
```
549    <S11:Envelope xmlns:S11="..." xmlns:wsse="...">
550       <S11:Header>
551             ...
552          <wsse:Security>
553             <wsse:UsernameToken>
554                 <wsse:Username>Zoe</wsse:Username>
555             </wsse:UsernameToken>
556          </wsse:Security>
557             ...
558       </S11:Header>
559       ...
560    </S11:Envelope>
```
561

562   **6.3 Binary Security Tokens**

563 **6.3.1 Attaching Security Tokens**

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

567 **6.3.2 Encoding Binary Security Tokens**

568 Binary security tokens (e.g., X.509 certificates and Kerberos [KERBEROS] tickets) or other non-
569 XML formats require a special encoding format for inclusion. This section describes a basic
570 framework for using binary security tokens. Subsequent specifications MUST describe the rules
571 for creating and processing specific binary security token formats.
572 The `<wsse:BinarySecurityToken>` element defines two attributes that are used to interpret
573 it. The `ValueType` attribute indicates what the security token is, for example, a Kerberos ticket.
574 The `EncodingType` tells how the security token is encoded, for example `Base64Binary`.
575 The following is an overview of the syntax:
576
```
577    <wsse:BinarySecurityToken wsu:Id=...
578                              EncodingType=...
579                              ValueType=.../>
```
580
581 The following describes the attributes and elements listed in the example above:
582 */wsse:BinarySecurityToken*
583   This element is used to include a binary-encoded security token.
584 */wsse:BinarySecurityToken/@wsu:Id*
585   An optional string label for this security token.
586 */wsse:BinarySecurityToken/@ValueType*
587   The `ValueType` attribute is used to indicate the "value space" of the encoded binary
588   data (e.g. an X.509 certificate). The `ValueType` attribute allows a URI that defines the
589   value type and space of the encoded binary data. Subsequent specifications MUST
590   define the `ValueType` value for the tokens that they define. The usage of `ValueType` is
591   RECOMMENDED.
592 */wsse:BinarySecurityToken/@EncodingType*
593   The `EncodingType` attribute is used to indicate, using a URI, the encoding format of the
594   binary data (e.g., `base64 encoded`). A new attribute is introduced, as there are issues

595	with the current schema validation tools that make derivations of mixed simple and
596	complex types difficult within XML Schema. The `EncodingType` attribute is interpreted
597	to indicate the encoding format of the element. The following encoding formats are pre-
598	defined (note that the URI fragments are relative to the URI for this specification):

599

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

600

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

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

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

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

613   6.4 XML Tokens

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

616 6.4.1 Identifying and Referencing Security Tokens

617 This specification also defines multiple mechanisms for identifying and referencing security
618 tokens using the `wsu:Id` attribute and the `<wsse:SecurityTokenReference>` element (as
619 well as some additional mechanisms). Please refer to the specific profile documents for the
620 appropriate reference mechanism. However, specific extensions MAY be made to the
621 `<wsse:SecurityTokenReference>` element.
622

7 Token References

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

7.1 SecurityTokenReference Element

626 A security token conveys a set of claims. Sometimes these claims reside somewhere else and
627 need to be "pulled" by the receiving application. The `<wsse:SecurityTokenReference>`
628 element provides an extensible mechanism for referencing security tokens.
629 The `<wsse:SecurityTokenReference>` element provides an open content model for
630 referencing security tokens because not all tokens support a common reference pattern.
631 Similarly, some token formats have closed schemas and define their own reference mechanisms.
632 The open content model allows appropriate reference mechanisms to be used when referencing
633 corresponding token types.
634 If a `<wsse:SecurityTokenReference>` is used outside of the `<wsse:Security>` header
635 block the meaning of the response and/or processing rules of the resulting references MUST be
636 specified by the containing element and are out of scope of this specification.
637 The following illustrates the syntax of this element:

638
```
639    <wsse:SecurityTokenReference wsu:Id="...">
640       ...
641    </wsse:SecurityTokenReference>
```
642
643 The following describes the elements defined above:
644 */wsse:SecurityTokenReference*
645   This element provides a reference to a security token.
646 */wsse:SecurityTokenReference/@wsu:Id*
647   A string label for this security token reference which names the reference. This attribute
648   does not indicate the ID of what is being referenced, that SHOULD be done using a
649   fragment URI in a `<wsse:Reference>` element within the
650   `<wsse:SecurityTokenReference>` element.
651 */wsse:SecurityTokenReference/@wsse:Usage*
652   This optional attribute is used to type the usage of the `<wsse:SecurityToken>`.
653   Usages are specified using URIs and multiple usages MAY be specified using XML list
654   semantics. No usages are defined by this specification.
655 */wsse:SecurityTokenReference/{any}*
656   This is an extensibility mechanism to allow different (extensible) types of security
657   references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
658   fault.
659 */wsse:SecurityTokenReference/@{any}*
660   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
661   added to the header. Unrecognized attributes SHOULD cause a fault.
662 All compliant implementations MUST be able to process a
663 `<wsse:SecurityTokenReference>` element.
664 This element can also be used as a direct child element of `<ds:KeyInfo>` to indicate a hint to
665 retrieve the key information from a security token placed somewhere else. In particular, it is
666 RECOMMENDED, when using XML Signature and XML Encryption, that a
667 `<wsse:SecurityTokenReference>` element be placed inside a `<ds:KeyInfo>` to reference
668 the security token used for the signature or encryption.
669 There are several challenges that implementations face when trying to interoperate. Processing
670 the IDs and references requires the recipient to *understand* the schema. This may be an
671 expensive task and in the general case impossible as there is no way to know the "schema
672 location" for a specific namespace URI. As well, the primary goal of a reference is to uniquely

673 identify the desired token. ID references are, by definition, unique by XML. However, other
674 mechanisms such as "principal name" are not required to be unique and therefore such
675 references may be not unique.
676 The following list provides a list of the specific reference mechanisms defined in WSS: SOAP
677 Message Security in preferred order (i.e., most specific to least specific):
678 •   **Direct References** – This allows references to included tokens using URI fragments and
679   external tokens using full URIs.
680 •   **Key Identifiers** – This allows tokens to be referenced using an opaque value that
681   represents the token (defined by token type/profile).
682 •   **Key Names** – This allows tokens to be referenced using a string that matches an identity
683   assertion within the security token. This is a subset match and may result in multiple
684   security tokens that match the specified name.
685 •   **Embedded References -** This allows tokens to be embedded (as opposed to a pointer
686   to a token that resides elsewhere).

687 **7.2 Direct References**

688 The `<wsse:Reference>` element provides an extensible mechanism for directly referencing
689 security tokens using URIs.
690 The following illustrates the syntax of this element:
691

```
692    <wsse:SecurityTokenReference wsu:Id="...">
693        <wsse:Reference URI="..." ValueType="..."/>
694    </wsse:SecurityTokenReference>
```

695
696 The following describes the elements defined above:
697 */wsse:SecurityTokenReference/wsse:Reference*
698   This element is used to identify an abstract URI location for locating a security token.
699 */wsse:SecurityTokenReference/wsse:Reference/@URI*
700   This optional attribute specifies an abstract URI for where to find a security token. If a
701   fragment is specified, then it indicates the local ID of the token being referenced.
702 */wsse:SecurityTokenReference/wsse:Reference/@ValueType*
703   This optional attribute specifies a URI that is used to identify the *type* of token being
704   referenced. This specification does not define any processing rules around the usage of
705   this attribute, however, specifications for individual token types MAY define specific
706   processing rules and semantics around the value of the URI and how it SHALL be
707   interpreted. If this attribute is not present, the URI MUST be processed as a normal URI.
708   The usage of `ValueType` is RECOMMENDED for references with local URIs.
709 */wsse:SecurityTokenReference/wsse:Reference/{any}*
710   This is an extensibility mechanism to allow different (extensible) types of security
711   references, based on a schema, to be passed. Unrecognized elements SHOULD cause a
712   fault.
713 */wsse:SecurityTokenReference/wsse:Reference/@{any}*
714   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
715   added to the header. Unrecognized attributes SHOULD cause a fault.
716 The following illustrates the use of this element:
717

```
718    <wsse:SecurityTokenReference
719            xmlns:wsse="...">
720       <wsse:Reference
721               URI="http://www.fabrikam123.com/tokens/Zoe"/>
722    </wsse:SecurityTokenReference>
```

723 **7.3 Key Identifiers**

724 Alternatively, if a direct reference is not used, then it is RECOMMENDED to use a key identifier to
725 specify/reference a security token instead of a `<ds:KeyName>`. A `KeyIdentifier` is a value
726 that can be used to uniquely identify a security token (e.g. a hash of the important elements of the
727 security token). The exact value type and generation algorithm varies by security token type (and
728 sometimes by the data within the token), Consequently, the values and algorithms are described
729 in the token-specific profiles rather than this specification.
730 The `<wsse:KeyIdentifier>` element SHALL be placed in the
731 `<wsse:SecurityTokenReference>` element to reference a token using an identifier. This
732 element SHOULD be used for all key identifiers.
733 The processing model assumes that the key identifier for a security token is constant.
734 Consequently, processing a key identifier is simply looking for a security token whose key
735 identifier matches a given specified constant.
736 The following is an overview of the syntax:
737

```
738    <wsse:SecurityTokenReference>
739       <wsse:KeyIdentifier wsu:Id="..."
740                           ValueType="..."
741                           EncodingType="...">
742          ...
743       </wsse:KeyIdentifier>
744    </wsse:SecurityTokenReference>
```

745
746 The following describes the attributes and elements listed in the example above:
747 /wsse:SecurityTokenReference/wsse:KeyIdentifier
748   This element is used to include a binary-encoded key identifier.
749 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@wsu:Id*
750   An optional string label for this identifier.
751 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@ValueType*
752   The optional `ValueType` attribute is used to indicate the type of `KeyIdentifier` being
753   used. Each specific token profile specifies the `KeyIdentifier` types that may be used
754   to refer to tokens of that type. It also specifies the critical semantics of the identifier, such
755   as whether the `KeyIdentifier` is unique to the key or the token. If no value is specified
756   then the key identifier will be interpreted in an application-specific manner.
757 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@EncodingType*
758   The optional `EncodingType` attribute is used to indicate, using a URI, the encoding
759   format of the `KeyIdentifier` (`#Base64Binary`). The base values defined in this
760   specification are used (Note that URI fragments are relative to this document's URI):
761

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

762
763 */wsse:SecurityTokenReference/wsse:KeyIdentifier/@{any}*
764   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
765   added.

766 **7.4 Embedded References**

767 In some cases a reference may be to an embedded token (as opposed to a pointer to a token
768 that resides elsewhere). To do this, the `<wsse:Embedded>` element is specified within a
769 `<wsse:SecurityTokenReference>` element.
770 The following is an overview of the syntax:
771

```
772        <wsse:SecurityTokenReference>
773           <wsse:Embedded wsu:Id="...">
774              ...
775           </wsse:Embedded>
776        </wsse:SecurityTokenReference>
```

778   The following describes the attributes and elements listed in the example above:
779   */wsse:SecurityTokenReference/wsse:Embedded*
780   This element is used to embed a token directly within a reference (that is, to create a
781   *local* or *literal* reference).
782   */wsse:SecurityTokenReference/wsse:Embedded/@wsu:Id*
783   An optional string label for this element. This allows this embedded token to be
784   referenced by a signature or encryption.
785   */wsse:SecurityTokenReference/wsse:Embedded/{any}*
786   This is an extensibility mechanism to allow any security token, based on schemas, to be
787   embedded. Unrecognized elements SHOULD cause a fault.
788   */wsse:SecurityTokenReference/wsse:Embedded/@{any}*
789   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
790   added. Unrecognized attributes SHOULD cause a fault.
791   The following example illustrates embedding a SAML assertion:

```
793        <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
794           <S11:Header>
795              <wsse:Security>
796                 ...
797                 <wsse:SecurityTokenReference>
798                    <wsse:Embedded wsu:Id="tok1">
799                       <saml:Assertion xmlns:saml="...">
800                          ...
801                       </saml:Assertion>
802                    </wsse:Embedded>
803                 </wsse:SecurityTokenReference>
804                 ...
805              <wsse:Security>
806           </S11:Header>
807           ...
808        </S11:Envelope>
```

809   7.5 ds:KeyInfo

810   The `<ds:KeyInfo>` element (from XML Signature) can be used for carrying the key information
811   and is allowed for different key types and for future extensibility. However, in this specification,
812   the use of `<wsse:BinarySecurityToken>` is the RECOMMENDED mechanism to carry key
813   material if the key type contains binary data. Please refer to the specific profile documents for the
814   appropriate way to carry key material.
815   The following example illustrates use of this element to fetch a named key:

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

820   7.6 Key Names

821   It is strongly RECOMMENDED to use `<wsse:KeyIdentifier>` elements. However, if key
822   names are used, then it is strongly RECOMMENDED that `<ds:KeyName>` elements conform to
823   the attribute names in section 2.3 of RFC 2253 (this is recommended by XML Signature for
824   `<ds:X509SubjectName>`) for interoperability.
825   Additionally, e-mail addresses, SHOULD conform to RFC 822:

826   `EmailAddress=ckaler@microsoft.com`
827

⁸²⁸ # 8 Signatures

829 Message producers may want to enable message recipients to determine whether a message
830 was altered in transit and to verify that the claims in a particular security token apply to the
831 producer of the message.
832 Demonstrating knowledge of a confirmation key associated with a token key-claim confirms the
833 accompanying token claims. Knowledge of a confirmation key may be demonstrated using that
834 key to create an XML Signature, for example. The relying party acceptance of the claims may
835 depend on its confidence in the token. Multiple tokens may contain a key-claim for a signature
836 and may be referenced from the signature using a `<wsse:SecurityTokenReference>`. A
837 key-claim may be an X.509 Certificate token, or a Kerberos service ticket token to give two
838 examples.
839 Because of the mutability of some SOAP headers, producers SHOULD NOT use the *Enveloped*
840 *Signature Transform* defined in XML Signature. Instead, messages SHOULD explicitly include
841 the elements to be signed. Similarly, producers SHOULD NOT use the *Enveloping Signature*
842 defined in XML Signature [XMLSIG].
843 This specification allows for multiple signatures and signature formats to be attached to a
844 message, each referencing different, even overlapping, parts of the message. This is important
845 for many distributed applications where messages flow through multiple processing stages. For
846 example, a producer may submit an order that contains an orderID header. The producer signs
847 the orderID header and the body of the request (the contents of the order). When this is received
848 by the order processing sub-system, it may insert a shippingID into the header. The order sub-
849 system would then sign, at a minimum, the orderID and the shippingID, and possibly the body as
850 well. Then when this order is processed and shipped by the shipping department, a shippedInfo
851 header might be appended. The shipping department would sign, at a minimum, the shippedInfo
852 and the shippingID and possibly the body and forward the message to the billing department for
853 processing. The billing department can verify the signatures and determine a valid chain of trust
854 for the order, as well as who authorized each step in the process.
855 All compliant implementations MUST be able to support the XML Signature standard.

856 ## 8.1 Algorithms

857 This specification builds on XML Signature and therefore has the same algorithm requirements as
858 those specified in the XML Signature specification.
859 The following table outlines additional algorithms that are strongly RECOMMENDED by this
860 specification:
861

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

862
863 As well, the following table outlines additional algorithms that MAY be used:

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

864
865 The Exclusive XML Canonicalization algorithm addresses the pitfalls of general canonicalization
866 that can occur from *leaky* namespaces with pre-existing signatures.

867 Finally, if a producer wishes to sign a message before encryption, then following the ordering
868 rules laid out in section 5, "Security Header", they SHOULD first prepend the signature element to
869 the `<wsse:Security>` header, and then prepend the encryption element, resulting in a
870 `<wsse:Security>` header that has the encryption element first, followed by the signature
871 element:
872

```
                <wsse:Security> header

                    [encryption element]
                    [signature element]
                              .
                              .
```
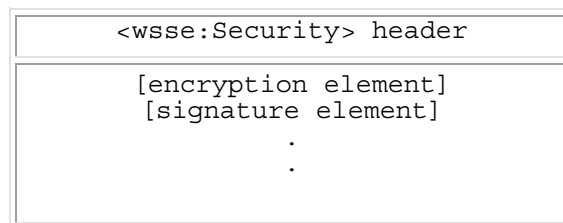
873
874 Likewise, if a producer wishes to sign a message after encryption, they SHOULD first prepend
875 the encryption element to the `<wsse:Security>` header, and then prepend the signature
876 element. This will result in a `<wsse:Security>` header that has the signature element first,
877 followed by the encryption element:
878

```
                <wsse:Security> header

                    [signature element]
                    [encryption element]
                              .
                              .
```
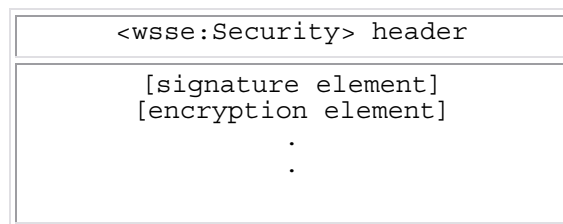
879
880 The XML Digital Signature WG has defined two canonicalization algorithms: XML
881 Canonicalization and Exclusive XML Canonicalization. To prevent confusion, the first is also
882 called Inclusive Canonicalization. Neither one solves all possible problems that can arise. The
883 following informal discussion is intended to provide guidance on the choice of which one to use
884 in particular circumstances. For a more detailed and technically precise discussion of these
885 issues see: [XML-C14N] and [EXC-C14N].
886 There are two problems to be avoided. On the one hand, XML allows documents to be changed
887 in various ways and still be considered equivalent. For example, duplicate namespace
888 declarations can be removed or created. As a result, XML tools make these kinds of changes
889 freely when processing XML. Therefore, it is vital that these equivalent forms match the same
890 signature.
891 On the other hand, if the signature simply covers something like xx:foo, its meaning may change
892 if xx is redefined. In this case the signature does not prevent tampering. It might be thought that
893 the problem could be solved by expanding all the values in line. Unfortunately, there are
894 mechanisms like XPATH which consider xx="http://example.com/"; to be different from
895 yy="http://example.com/"; even though both xx and yy are bound to the same namespace.
896 The fundamental difference between the Inclusive and Exclusive Canonicalization is the
897 namespace declarations which are placed in the output. Inclusive Canonicalization copies all the
898 declarations that are currently in force, even if they are defined outside of the scope of the
899 signature. It also copies any xml: attributes that are in force, such as `xml:lang` or `xml:base`.
900 This guarantees that all the declarations you might make use of will be unambiguously specified.
901 The problem with this is that if the signed XML is moved into another XML document which has
902 other declarations, the Inclusive Canonicalization will copy then and the signature will be invalid.
903 This can even happen if you simply add an attribute in a different namespace to the surrounding
904 context.
905 Exclusive Canonicalization tries to figure out what namespaces you are actually using and just
906 copies those. Specifically, it copies the ones that are "visibly used", which means the ones that

907 are a part of the XML syntax. However, it does not look into attribute values or element content,
908 so the namespace declarations required to process these are not copied. For example
909 if you had an attribute like xx:foo="yy:bar" it would copy the declaration for xx, but not yy. (This
910 can even happen without your knowledge because XML processing tools will add `xsi:type` if
911 you use a schema subtype.) It also does not copy the xml: attributes that are declared outside the
912 scope of the signature.
913 Exclusive Canonicalization allows you to create a list of the namespaces that must be declared,
914 so that it will pick up the declarations for the ones that are not visibly used. The only problem is
915 that the software doing the signing must know what they are. In a typical SOAP software
916 environment, the security code will typically be unaware of all the namespaces being used by
917 the application in the message body that it is signing.
918 Exclusive Canonicalization is useful when you have a signed XML document that you wish to
919 insert into other XML documents. A good example is a signed SAML assertion which might be
920 inserted as a XML Token in the security header of various SOAP messages. The Issuer who
921 signs the assertion will be aware of the namespaces being used and able to construct the list.
922 The use of Exclusive Canonicalization will insure the signature verifies correctly every time.
923 Inclusive Canonicalization is useful in the typical case of signing part or all of the SOAP body in
924 accordance with this specification. This will insure all the declarations fall under the signature,
925 even though the code is unaware of what namespaces are being used. At the same time, it is
926 less likely that the signed data (and signature element) will be inserted in some other XML
927 document. Even if this is desired, it still may not be feasible for other reasons, for example there
928 may be Id's with the same value defined in both XML documents.
929 In other situations it will be necessary to study the requirements of the application and the
930 detailed operation of the canonicalization methods to determine which is appropriate.
931 This section is non-normative.
932

8.2 Signing Messages

934 The `<wsse:Security>` header block MAY be used to carry a signature compliant with the XML
935 Signature specification within a SOAP Envelope for the purpose of signing one or more elements
936 in the SOAP Envelope. Multiple signature entries MAY be added into a single SOAP Envelope
937 within one `<wsse:Security>` header block. Producers SHOULD sign all important elements of
938 the message, and careful thought must be given to creating a signing policy that requires signing
939 of parts of the message that might legitimately be altered in transit.
940 SOAP applications MUST satisfy the following conditions:
941 A compliant implementation MUST be capable of processing the required elements defined in the
942 XML Signature specification.
943 To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element
944 conforming to the XML Signature specification MUST be prepended to the existing content of the
945 `<wsse:Security>` header block, in order to indicate to the receiver the correct order of
946 operations. All the `<ds:Reference>` elements contained in the signature SHOULD refer to a
947 resource within the enclosing SOAP envelope as described in the XML Signature specification.
948 However, since the SOAP message exchange model allows intermediate applications to modify
949 the Envelope (add or delete a header block; for example), XPath filtering does not always result
950 in the same objects after message delivery. Care should be taken in using XPath filtering so that
951 there is no subsequent validation failure due to such modifications.
952 The problem of modification by intermediaries (especially active ones) is applicable to more than
953 just XPath processing. Digital signatures, because of canonicalization and digests, present
954 particularly fragile examples of such relationships. If overall message processing is to remain
955 robust, intermediaries must exercise care that the transformation algorithms used do not affect
956 the validity of a digitally signed component.
957 Due to security concerns with namespaces, this specification strongly RECOMMENDS the use of
958 the "Exclusive XML Canonicalization" algorithm or another canonicalization algorithm that
959 provides equivalent or greater protection.

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

962 **8.3 Signing Tokens**

963 It is often desirable to sign security tokens that are included in a message or even external to the
964 message. The XML Signature specification provides several common ways for referencing
965 information to be signed such as URIs, IDs, and XPath, but some token formats may not allow
966 tokens to be referenced using URIs or IDs and XPaths may be undesirable in some situations.
967 This specification allows different tokens to have their own unique reference mechanisms which
968 are specified in their profile as extensions to the `<wsse:SecurityTokenReference>` element.
969 This element provides a uniform referencing mechanism that is guaranteed to work with all token
970 formats. Consequently, this specification defines a new reference option for XML Signature: the
971 STR Dereference Transform.
972 This transform is specified by the URI `#STR-Transform` (Note that URI fragments are relative to
973 this document's URI) and when applied to a `<wsse:SecurityTokenReference>` element it
974 means that the output is the token referenced by the `<wsse:SecurityTokenReference>`
975 element not the element itself.
976 As an overview the processing model is to echo the input to the transform except when a
977 `<wsse:SecurityTokenReference>` element is encountered. When one is found, the element
978 is not echoed, but instead, it is used to locate the token(s) matching the criteria and rules defined
979 by the `<wsse:SecurityTokenReference>` element and echo it (them) to the output.
980 Consequently, the output of the transformation is the resultant sequence representing the input
981 with any `<wsse:SecurityTokenReference>` elements replaced by the referenced security
982 token(s) matched.
983 The following illustrates an example of this transformation which references a token contained
984 within the message envelope:
985

```
986    ...
987    <wsse:SecurityTokenReference wsu:Id="Str1">
988          ...
989    </wsse:SecurityTokenReference>
990    ...
991    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
992         <ds:SignedInfo>
993          ...
994           <ds:Reference URI="#Str1">
995             <ds:Transforms>
996               <ds:Transform
997                   Algorithm="...#STR-Transform">
998                 <wsse:TransformationParameters>
999                   <ds:CanonicalizationMethod
1000                        Algorithm="http://www.w3.org/TR/2001/REC-xml-
1001   c14n-20010315" />
1002               </wsse:TransformationParameters>
1003             </ds:Transform>
1004           <ds:DigestMethod Algorithm=
1005                             "http://www.w3.org/2000/09/xmldsig#sha1"/>
1006           <ds:DigestValue>...</ds:DigestValue>
1007         </ds:Reference>
1008       </ds:SignedInfo>
1009       <ds:SignatureValue></ds:SignatureValue>
1010    </ds:Signature>
1011    ...
```
1012
1013 The following describes the attributes and elements listed in the example above:
1014 */wsse:TransformationParameters*

| 1015 | This element is used to wrap parameters for a transformation allows elements even from |
| 1016 | the XML Signature namespace. |

/wsse:TransformationParameters/ds:Canonicalization
1017
1018   This specifies the canolicalization algorithm to apply to the selected data.

/wsse:TransformationParameters/{any}
1019
1020   This is an extensibility mechanism to allow different (extensible) parameters to be
1021   specified in the future. Unrecognized parameters SHOULD cause a fault.

/wsse:TransformationParameters/@{any}
1022
1023   This is an extensibility mechanism to allow additional attributes, based on schemas, to be
1024   added to the element in the future. Unrecognized attributes SHOULD cause a fault.

1025

1026 The following is a detailed specification of the transformation.
1027 The algorithm is identified by the URI: #STR-Transform
1028 Transform Input:
1029   •   The input is a node set. If the input is an octet stream, then it is automatically parsed; cf.
1030   XML Digital Signature [XMLSIG].
1031 Transform Output:
1032   •   The output is an octet steam.
1033 Syntax:
1034   •   The transform takes a single mandatory parameter, a
1035   `<ds:CanonicalizationMethod>` element, which is used to serialize the input node
1036   set. Note, however, that the output may not be strictly in canonical form, per the
1037   canonicalization algorithm; however, the output is canonical, in the sense that it is
1038   unambiguous. However, because of syntax requirements in the XML Signature
1039   definition, this parameter MUST be wrapped in a
1040   `<wsse:TransformationParameters>` element.
1041 Processing Rules:
1042   •   Let N be the input node set.
1043   •   Let R be the set of all `<wsse:SecurityTokenReference>` elements in N.
1044   •   For each Ri in R, let Di be the result of dereferencing Ri.
1045   •   If Di cannot be determined, then the transform MUST signal a failure.
1046   •   If Di is an XML security token (e.g., a SAML assertion or a
1047   `<wsse:BinarySecurityToken>` element), then let Ri' be Di.Otherwise, Di is a raw
1048   binary security token; i.e., an octet stream. In this case, let Ri' be a node set consisting of
1049   a `<wsse:BinarySecurityToken>` element, utilizing the same namespace prefix as
1050   the `<wsse:SecurityTokenReference>` element Ri, with no `EncodingType` attribute,
1051   a `ValueType` attribute identifying the content of the security token, and text content
1052   consisting of the binary-encoded security token, with no white space.
1053   •   Finally, employ the canonicalization method specified as a parameter to the transform to
1054   serialize N to produce the octet stream output of this transform; but, in place of any
1055   dereferenced `<wsse:SecurityTokenReference>` element Ri and its descendants,
1056   process the dereferenced node set Ri' instead. During this step, canonicalization of the
1057   replacement node set MUST be augmented as follows:
1058   ○   Note: A namespace declaration `xmlns=""` MUST be emitted with every apex
1059   element that has no namespace node declaring a value for the default
1060   namespace; cf. XML Decryption Transform.

1061   **8.4 Signature Validation**

1062 The validation of a `<ds:Signature>` element inside an `<wsse:Security>` header block
1063 SHALL fail if:
1064   •   the syntax of the content of the element does not conform to this specification, or
1065   •   the validation of the signature contained in the element fails according to the core
1066   validation of the XML Signature specification [XMLSIG], or

1067 • the application applying its own validation policy rejects the message for some reason
1068 (e.g., the signature is created by an untrusted key – verifying the previous two steps only
1069 performs cryptographic validation of the signature).
1070 If the validation of the signature element fails, applications MAY report the failure to the producer
1071 using the fault codes defined in Section 12 Error Handling.

1072 **8.5 Example**

1073 The following sample message illustrates the use of integrity and security tokens. For this
1074 example, only the message body is signed.
1075

```
1076    <?xml version="1.0" encoding="utf-8"?>
1077    <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1078    xmlns:ds="...">
1079        <S11:Header>
1080            <wsse:Security>
1081                <wsse:BinarySecurityToken
1082                        ValueType="...#X509v3"
1083                        EncodingType="...#Base64Binary"
1084                        wsu:Id="X509Token">
1085                    MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1086                </wsse:BinarySecurityToken>
1087                <ds:Signature>
1088                    <ds:SignedInfo>
1089                        <ds:CanonicalizationMethod Algorithm=
1090                            "http://www.w3.org/2001/10/xml-exc-c14n#"/>
1091                        <ds:SignatureMethod Algorithm=
1092                            "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
1093                        <ds:Reference URI="#myBody">
1094                            <ds:Transforms>
1095                                <ds:Transform Algorithm=
1096                                    "http://www.w3.org/2001/10/xml-exc-c14n#"/>
1097                            </ds:Transforms>
1098                            <ds:DigestMethod Algorithm=
1099                                "http://www.w3.org/2000/09/xmldsig#sha1"/>
1100                            <ds:DigestValue>EULddytSo1...</ds:DigestValue>
1101                        </ds:Reference>
1102                    </ds:SignedInfo>
1103                    <ds:SignatureValue>
1104                        BL8jdfToEb1l/vXcMZNNjPOV...
1105                    </ds:SignatureValue>
1106                    <ds:KeyInfo>
1107                        <wsse:SecurityTokenReference>
1108                            <wsse:Reference URI="#X509Token"/>
1109                        </wsse:SecurityTokenReference>
1110                    </ds:KeyInfo>
1111                </ds:Signature>
1112            </wsse:Security>
1113        </S11:Header>
1114        <S11:Body wsu:Id="myBody">
1115            <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
1116                QQQ
1117            </tru:StockSymbol>
1118        </S11:Body>
1119    </S11:Envelope>
```

9 Encryption

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

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

All compliant implementations MUST be able to support the XML Encryption standard [XMLENC].

9.1 xenc:ReferenceList

The `<xenc:ReferenceList>` element from XML Encryption [XMLENC] MAY be used to create a manifest of encrypted portion(s), which are expressed as `<xenc:EncryptedData>` elements within the envelope. An element or element content to be encrypted by this encryption step MUST be replaced by a corresponding `<xenc:EncryptedData>` according to XML Encryption. All the `<xenc:EncryptedData>` elements created by this encryption step SHOULD be listed in `<xenc:DataReference>` elements inside one or more `<xenc:ReferenceList>` element.

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

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

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:ds="..." xmlns:xenc="...">
    <S11:Header>
        <wsse:Security>
            <xenc:ReferenceList>
                <xenc:DataReference URI="#bodyID"/>
            </xenc:ReferenceList>
        </wsse:Security>
    </S11:Header>
    <S11:Body>
        <xenc:EncryptedData Id="bodyID">
          <ds:KeyInfo>
            <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
          </ds:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
          </xenc:CipherData>
```

```
1171              </xenc:EncryptedData>
1172          </S11:Body>
1173      </S11:Envelope>
```

9.2 xenc:EncryptedKey

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

```
1187      <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1188      xmlns:ds="..." xmlns:xenc="...">
1189          <S11:Header>
1190              <wsse:Security>
1191                  <xenc:EncryptedKey>
1192                    ...
1193                    <ds:KeyInfo>
1194                      <wsse:SecurityTokenReference>
1195                        <ds:X509IssuerSerial>
1196                          <ds:X509IssuerName>
1197                            DC=ACMECorp, DC=com
1198                          </ds:X509IssuerName>
1199      <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
1200                        </ds:X509IssuerSerial>
1201                      </wsse:SecurityTokenReference>
1202                    </ds:KeyInfo>
1203                    ...
1204                  </xenc:EncryptedKey>
1205        ...
1206              </wsse:Security>
1207          </S11:Header>
1208          <S11:Body>
1209              <xenc:EncryptedData Id="bodyID">
1210                  <xenc:CipherData>
1211                    <xenc:CipherValue>...</xenc:CipherValue>
1212                  </xenc:CipherData>
1213              </xenc:EncryptedData>
1214          </S11:Body>
1215      </S11:Envelope>
```

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

9.3 Processing Rules

Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the
XML Encryption specification. An encrypted SOAP envelope MUST still be a valid SOAP
envelope. The message creator MUST NOT encrypt the `<S11:Envelope>`,
`<S12:Envelope>`,, `<S11:Header>`, `<S12:Header>`, or `<S11:Body>`, `<S12:Body>`
elements but MAY encrypt child elements of either the `<S11:Header>`, `<S12:Header>` and

1226   `<S11:Body>` or `<S12:Body>` elements. Multiple steps of encryption MAY be added into a
1227   single `<wsse:Security>` header block if they are targeted for the same recipient.
1228   When an element or element content inside a SOAP envelope (e.g. the contents of the
1229   `<S11:Body>` or `<S12:Body>` elements) are to be encrypted, it MUST be replaced by an
1230   `<xenc:EncryptedData>`, according to XML Encryption and it SHOULD be referenced from the
1231   `<xenc:ReferenceList>` element created by this encryption step.

1232   9.3.1 Encryption

1233   The general steps (non-normative) for creating an encrypted SOAP message in compliance with
1234   this specification are listed below (note that use of `<xenc:ReferenceList>` is
1235   RECOMMENDED).

- 1236   • Create a new SOAP envelope.
- 1237   • Create a `<wsse:Security>` header
- 1238   • When an `<xenc:EncryptedKey>` is used, create a `<xenc:EncryptedKey>` sub-
  1239   element of the `<wsse:Security>` element. This `<xenc:EncryptedKey>` sub-
  1240   element SHOULD contain an `<xenc:ReferenceList>` sub-element, containing a
  1241   `<xenc:DataReference>` to each `<xenc:EncryptedData>` element that was
  1242   encrypted using that key.
- 1243   • Locate data items to be encrypted, i.e., XML elements, element contents within the target
  1244   SOAP envelope.
- 1245   • Encrypt the data items as follows: For each XML element or element content within the
  1246   target SOAP envelope, encrypt it according to the processing rules of the XML
  1247   Encryption specification [XMLENC]. Each selected original element or element content
  1248   MUST be removed and replaced by the resulting `<xenc:EncryptedData>` element.
- 1249   • The optional `<ds:KeyInfo>` element in the `<xenc:EncryptedData>` element MAY
  1250   reference another `<ds:KeyInfo>` element. Note that if the encryption is based on an
  1251   attached security token, then a `<wsse:SecurityTokenReference>` element SHOULD
  1252   be added to the `<ds:KeyInfo>` element to facilitate locating it.
- 1253   • Create an `<xenc:DataReference>` element referencing the generated
  1254   `<xenc:EncryptedData>` elements. Add the created `<xenc:DataReference>`
  1255   element to the `<xenc:ReferenceList>`.
- 1256   • Copy all non-encrypted data.

1257   9.3.2 Decryption

1258   On receiving a SOAP envelope containing encryption header elements, for each encryption
1259   header element the following general steps should be processed (non-normative):
1260   Identify any decryption keys that are in the recipient's possession, then identifying any message
1261   elements that it is able to decrypt.
1262   Locate the `<xenc:EncryptedData>` items to be decrypted (possibly using the
1263   `<xenc:ReferenceList>`).
1264   Decrypt them as follows:
1265   For each element in the target SOAP envelope, decrypt it according to the processing rules of the
1266   XML Encryption specification and the processing rules listed above.
1267   If the decryption fails for some reason, applications MAY report the failure to the producer using
1268   the fault code defined in Section 12 Error Handling of this specification.
1269   It is possible for overlapping portions of the SOAP message to be encrypted in such a way that
1270   they are intended to be decrypted by SOAP nodes acting in different Roles. In this case, the
1271   `<xenc:ReferenceList>` or `<xenc:EncryptedKey>` elements identifying these encryption
1272   operations will necessarily appear in different `<wsse:Security>` headers. Since SOAP does
1273   not provide any means of specifying the order in which different Roles will process their
1274   respective headers, this order is not specified by this specification and can only be determined by
1275   a prior agreement.

9.4 Decryption Transformation

1277 The ordering semantics of the `<wsse:Security>` header are sufficient to determine if
1278 signatures are over encrypted or unencrypted data. However, when a signature is included in
1279 one `<wsse:Security>` header and the encryption data is in another `<wsse:Security>`
1280 header, the proper processing order may not be apparent.
1281 If the producer wishes to sign a message that MAY subsequently be encrypted by an
1282 intermediary then the producer MAY use the Decryption Transform for XML Signature to explicitly
1283 specify the order of decryption.
1284

1285 10 Security Timestamps

1286 It is often important for the recipient to be able to determine the *freshness* of security semantics.
1287 In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.
1288 This specification does not provide a mechanism for synchronizing time. The assumption is that
1289 time is trusted or additional mechanisms, not described here, are employed to prevent replay.
1290 This specification defines and illustrates time references in terms of the `xsd:dateTime` type
1291 defined in XML Schema. It is RECOMMENDED that all time references use this type. It is further
1292 RECOMMENDED that all references be in UTC time. Implementations MUST NOT generate time
1293 instants that specify leap seconds. If, however, other time types are used, then the `ValueType`
1294 attribute (described below) MUST be specified to indicate the data type of the time format.
1295 Requestors and receivers SHOULD NOT rely on other applications supporting time resolution
1296 finer than milliseconds.
1297 The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and
1298 expiration times of the security semantics in a message.
1299 All times MUST be in UTC format as specified by the XML Schema type (dateTime). It should be
1300 noted that times support time precision as defined in the XML Schema specification.
1301 The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and
1302 may only be present at most once per header (that is, per SOAP actor/role).
1303 The ordering within the element is as illustrated below. The ordering of elements in the
1304 `<wsu:Timestamp>` element is fixed and MUST be preserved by intermediaries.
1305 The schema outline for the `<wsu:Timestamp>` element is as follows:
1306
```
1307    <wsu:Timestamp wsu:Id="...">
1308        <wsu:Created ValueType="...">...</wsu:Created>
1309        <wsu:Expires  ValueType="...">...</wsu:Expires>
1310        ...
1311    </wsu:Timestamp>
```
1312

1313 The following describes the attributes and elements listed in the schema above:
1314 */wsu:Timestamp*
1315   This is the element for indicating message timestamps.
1316 */wsu:Timestamp/wsu:Created*
1317   This represents the creation time of the security semantics. This element is optional, but
1318   can only be specified once in a `<wsu:Timestamp>` element. Within the SOAP
1319   processing model, creation is the instant that the infoset is serialized for transmission.
1320   The creation time of the message SHOULD NOT differ substantially from its transmission
1321   time. The difference in time should be minimized.
1322 */wsu:Timestamp/wsu:Expires*
1323   This element represents the expiration of the security semantics. This is optional, but
1324   can appear at most once in a `<wsu:Timestamp>` element. Upon expiration, the
1325   requestor asserts that its security semantics are no longer valid. It is strongly
1326   RECOMMENDED that recipients (anyone who processes this message) discard (ignore)
1327   any message whose security semantics have passed their expiration. A Fault code
1328   (`wsu:MessageExpired`) is provided if the recipient wants to inform the requestor that its
1329   security semantics were expired. A service MAY issue a Fault indicating the security
1330   semantics have expired.
1331 */wsu:Timestamp/{any}*
1332   This is an extensibility mechanism to allow additional elements to be added to the
1333   element. Unrecognized elements SHOULD cause a fault.
1334 */wsu:Timestamp/@wsu:Id*

1335   This optional attribute specifies an XML Schema ID that can be used to reference this
1336   element (the timestamp). This is used, for example, to reference the timestamp in a XML
1337   Signature.
1338 */wsu:Timestamp/@{any}*
1339   This is an extensibility mechanism to allow additional attributes to be added to the
1340   element. Unrecognized attributes SHOULD cause a fault.
1341 The expiration is relative to the requestor's clock. In order to evaluate the expiration time,
1342 recipients need to recognize that the requestor's clock may not be synchronized to the recipient's
1343 clock. The recipient, therefore, MUST make an assessment of the level of trust to be placed in
1344 the requestor's clock, since the recipient is called upon to evaluate whether the expiration time is
1345 in the past relative to the requestor's, not the recipient's, clock. The recipient may make a
1346 judgment of the requestor's likely current clock time by means not described in this specification,
1347 for example an out-of-band clock synchronization protocol. The recipient may also use the
1348 creation time and the delays introduced by intermediate SOAP roles to estimate the degree of
1349 clock skew.
1350 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.
1351
1352
```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
  <S11:Header>
    <wsse:Security>
      <wsu:Timestamp wsu:Id="timestamp">
          <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
          <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
      </wsu:Timestamp>
      ...
    </wsse:Security>
    ...
  </S11:Header>
  <S11:Body>
      ...
  </S11:Body>
</S11:Envelope>
```

11 Extended Example

1368 The following sample message illustrates the use of security tokens, signatures, and encryption.
1369 For this example, the timestamp and the message body are signed prior to encryption. The
1370 decryption transformation is not needed as the signing/encryption order is specified within the
1371 `<wsse:Security>` header.
1372

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
xmlns:xenc="..." xmlns:ds="...">
(003)    <S11:Header>
(004)       <wsse:Security>
(005)          <wsu:Timestamp wsu:Id="T0">
(006)             <wsu:Created>
(007)                   2001-09-13T08:42:00Z</wsu:Created>
(008)          </wsu:Timestamp>
(009)
(010)          <wsse:BinarySecurityToken
                     ValueType="...#X509v3"
                     wsu:Id="X509Token"
                     EncodingType="...#Base64Binary">
(011)          MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
(012)          </wsse:BinarySecurityToken>
(013)          <xenc:EncryptedKey>
(014)             <xenc:EncryptionMethod Algorithm=
                        "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
(015)             <ds:KeyInfo>
(016)                <wsse:KeyIdentifier
                        EncodingType="...#Base64Binary"
                     ValueType="...#X509v3">MIGfMa0GCSq...
(017)                </wsse:KeyIdentifier>
(018)             </ds:KeyInfo>
(019)             <xenc:CipherData>
(020)                <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(021)                </xenc:CipherValue>
(022)             </xenc:CipherData>
(023)             <xenc:ReferenceList>
(024)                <xenc:DataReference URI="#enc1"/>
(025)             </xenc:ReferenceList>
(026)          </xenc:EncryptedKey>
(027)          <ds:Signature>
(028)             <ds:SignedInfo>
(029)                <ds:CanonicalizationMethod
                     Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(030)                <ds:SignatureMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(031)                <ds:Reference URI="#T0">
(032)                   <ds:Transforms>
(033)                      <ds:Transform
                     Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(034)                   </ds:Transforms>
(035)                   <ds:DigestMethod
                      Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(036)                   <ds:DigestValue>LyLsF094hPi4wPU...
(037)                   </ds:DigestValue>
(038)                </ds:Reference>
(039)                <ds:Reference URI="#body">
(040)                   <ds:Transforms>
(041)                      <ds:Transform
```

```
1425                               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1426     (042)                    </ds:Transforms>
1427     (043)                    <ds:DigestMethod
1428                          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
1429     (044)                    <ds:DigestValue>LyLsF094hPi4wPU...
1430     (045)                    </ds:DigestValue>
1431     (046)                </ds:Reference>
1432     (047)            </ds:SignedInfo>
1433     (048)            <ds:SignatureValue>
1434     (049)                    Hp1ZkmFZ/2kQLXDJbchm5gK...
1435     (050)            </ds:SignatureValue>
1436     (051)            <ds:KeyInfo>
1437     (052)                <wsse:SecurityTokenReference>
1438     (053)                    <wsse:Reference URI="#X509Token"/>
1439     (054)                </wsse:SecurityTokenReference>
1440     (055)            </ds:KeyInfo>
1441     (056)        </ds:Signature>
1442     (057)    </wsse:Security>
1443     (058)  </S11:Header>
1444     (059)  <S11:Body wsu:Id="body">
1445     (060)        <xenc:EncryptedData
1446                      Type="http://www.w3.org/2001/04/xmlenc#Element"
1447                      wsu:Id="enc1">
1448     (061)          <xenc:EncryptionMethod
1449                  Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
1450      cbc"/>
1451     (062)          <xenc:CipherData>
1452     (063)              <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1453     (064)              </xenc:CipherValue>
1454     (065)          </xenc:CipherData>
1455     (066)        </xenc:EncryptedData>
1456     (067)    </S11:Body>
1457     (068) </S11:Envelope>
```

1459 Let's review some of the key sections of this example:
1460 Lines (003)-(058) contain the SOAP message headers.
1461 Lines (004)-(057) represent the `<wsse:Security>` header block. This contains the security-
1462 related information for the message.
1463 Lines (005)-(008) specify the timestamp information. In this case it indicates the creation time of
1464 the security semantics.
1465 Lines (010)-(012) specify a security token that is associated with the message. In this case, it
1466 specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64
1467 encoding of the certificate.
1468 Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a
1469 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to
1470 encrypt the key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the
1471 symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines
1472 (023)-(025) identify the encryption block in the message that uses this symmetric key. In this
1473 case it is only used to encrypt the body (Id="enc1").
1474 Lines (027)-(056) specify the digital signature. In this example, the signature is based on the
1475 X.509 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039)
1476 references the message body.
1477 Lines (048)-(050) indicate the actual signature value – specified in Line (043).
1478 Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the X.509
1479 certificate included in the message. Line (053) provides a URI link to the Lines (010)-(012).
1480 The body of the message is represented by Lines (059)-(067).
1481 Lines (060)-(066) represent the encrypted metadata and form of the body using XML Encryption.
1482 Line (060) indicates that the "element value" is being replaced and identifies this encryption. Line
1483 (061) specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the

1484 actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the
1485 key as the key references this encryption – Line (024).

1486 12 Error Handling

1487 There are many circumstances where an *error* can occur while processing security information.
1488 For example:
1489 • Invalid or unsupported type of security token, signing, or encryption
1490 • Invalid or unauthenticated or unauthenticatable security token
1491 • Invalid signature
1492 • Decryption failure
1493 • Referenced security token is unavailable
1494 • Unsupported namespace

1495 If a service does not perform its normal operation because of the contents of the Security header,
1496 then that MAY be reported using SOAP's Fault Mechanism. This specification does not mandate
1497 that faults be returned as this could be used as part of a denial of service or cryptographic
1498 attack. We combine signature and encryption failures to mitigate certain types of attacks.
1499 If a failure is returned to a producer then the failure MUST be reported using the SOAP Fault
1500 mechanism. The following tables outline the predefined security fault codes. The "unsupported"
1501 classes of errors are as follows. Note that the reason text provided below is RECOMMENDED,
1502 but alternative text MAY be provided if more descriptive or preferred by the implementation. The
1503 tables below are defined in terms of SOAP 1.1. For SOAP 1.2, the Fault/Code/Value is
1504 `env:Sender` (as defined in SOAP 1.2) and the Fault/Code/Subcode/Value is the *faultcode* below
1505 and the Fault/Reason/Text is the *faultstring* below.
1506

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

1507 The "failure" class of errors are:

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

13 Security Considerations

As stated in the Goals and Requirements section of this document, this specification is meant to provide extensible framework and flexible syntax, with which one could implement various security mechanisms. This framework and syntax by itself *does not provide any guarantee of security.* When implementing and using this framework and syntax, one must make every effort to ensure that the result is not vulnerable to any one of a wide range of attacks.

13.1 General Considerations

It is not feasible to provide a comprehensive list of security considerations for such an extensible set of mechanisms. A complete security analysis MUST be conducted on specific solutions based on this specification. Below we illustrate some of the security concerns that often come up with protocols of this type, but we stress that this *is not an exhaustive list of concerns.*

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

There are other security concerns that one may need to consider in security protocols. The list above should not be used as a "check list" instead of a comprehensive security analysis. The next section will give a few details on some of the considerations in this list.

13.2 Additional Considerations

13.2.1 Replay

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

13.2.2 Combining Security Mechanisms

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

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

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

13.2.3 Challenges

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

13.2.4 Protecting Security Tokens and Keys

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

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

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

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

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

1602 **13.2.5 Protecting Timestamps and Ids**

1603 In order to *trust* `wsu:Id` attributes and `<wsu:Timestamp>` elements, they SHOULD be signed
1604 using the mechanisms outlined in this specification. This allows readers of the IDs and
1605 timestamps information to be certain that the IDs and timestamps haven't been forged or altered
1606 in any way. It is strongly RECOMMENDED that IDs and timestamp elements be signed.
1607
1608 This section is non-normative.

14 Interoperability Notes

1609

1610 Based on interoperability experiences with this and similar specifications, the following list
1611 highlights several common areas where interoperability issues have been discovered. Care
1612 should be taken when implementing to avoid these issues. It should be noted that some of these
1613 may seem "obvious", but have been problematic during testing.

1614 • **Key Identifiers:** Make sure you understand the algorithm and how it is applied to security
1615 tokens.

1616 • **EncryptedKey:** The `<xenc:EncryptedKey>` element from XML Encryption requires a
1617 Type attribute whose value is one of a pre-defined list of values. Ensure that a correct
1618 value is used.

1619 • **Encryption Padding:** The XML Encryption random block cipher padding has caused
1620 issues with certain decryption implementations; be careful to follow the specifications
1621 exactly.

1622 • **IDs:** The specification recognizes three specific ID elements: the global wsu:Id attribute
1623 and the local Id attributes on XML Signature and XML Encryption elements (because the
1624 latter two do not allow global attributes). If any other element does not allow global
1625 attributes, it cannot be directly signed using an ID reference. Note that the global
1626 attribute `wsu:Id` MUST carry the namespace specification.

1627 • **Time Formats:** This specification uses a restricted version of the XML Schema
1628 `xsd:dateTime` element. Take care to ensure compliance with the specified restrictions.

1629 • **Byte Order Marker (BOM):** Some implementations have problems processing the BOM
1630 marker. It is suggested that usage of this be optional.

1631 • **SOAP, WSDL, HTTP:** Various interoperability issues have been seen with incorrect
1632 SOAP, WSDL, and HTTP semantics being applied. Care should be taken to carefully
1633 adhere to these specifications and any interoperability guidelines that are available.

1634 This section is non-normative.

15 Privacy Considerations

1635

1636 In the context of this specification, we are only concerned with potential privacy violation by the
1637 security elements defined here. Privacy of the content of the payload message is out of scope.
1638 Producers or sending applications should be aware that claims, as collected in security tokens,
1639 are typically personal information, and should thus only be sent according to the producer's
1640 privacy policies. Future standards may allow privacy obligations or restrictions to be added to this
1641 data. Unless such standards are used, the producer must ensure by out-of-band means that the
1642 recipient is bound to adhering to all restrictions associated with the data, and the recipient must
1643 similarly ensure by out-of-band means that it has the necessary consent for its intended
1644 processing of the data.
1645 If claim data are visible to intermediaries, then the policies must also allow the release to these
1646 intermediaries. As most personal information cannot be released to arbitrary parties, this will
1647 typically require that the actors are referenced in an identifiable way; such identifiable references
1648 are also typically needed to obtain appropriate encryption keys for the intermediaries.
1649 If intermediaries add claims, they should be guided by their privacy policies just like the original
1650 producers.
1651 Intermediaries may also gain traffic information from a SOAP message exchange, e.g., who
1652 communicates with whom at what time. Producers that use intermediaries should verify that
1653 releasing this traffic information to the chosen intermediaries conforms to their privacy policies.
1654 This section is non-normative.

1655 16 References

| 1656 | **[GLOSS]** | Informational RFC 2828, "Internet Security Glossary," May 2000. |

| 1657 1658 | **[KERBEROS]** | J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993, http://www.ietf.org/rfc/rfc1510.txt . |

| 1659 1660 | **[KEYWORDS]** | S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997 |

| 1661 1662 1663 | **[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 |

| 1664 | **[SOAP11]** | W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000. |

| 1665 1666 | **[SOAP12]** | W3C Recomendation**, "**http://www.w3.org/TR/2003/REC-soap12-part1-20030624/", 24 June 2003 |

| 1667 1668 | **[SOAPSEC]** | W3C Note, "SOAP Security Extensions: Digital Signature," 06 February 2001. |

| 1669 1670 1671 | **[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. |

| 1672 | **[XPATH]** | W3C Recommendation, "XML Path Language", 16 November 1999 |

1673 The following are non-normative references included for background and related material:

| 1674 1675 1676 | **[WS-SECURITY]** | "Web Services Security Language", IBM, Microsoft, VeriSign, April 2002. "WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002. "WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002. |

| 1677 | **[XMLC14N]** | W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001 |

| 1678 1679 | **[EXCC14N]** | W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8 July 2002. |

| 1680 1681 | **[XMLENC]** | W3C Working Draft, "XML Encryption Syntax and Processing," 04 March 2002 |

| 1682 1683 | | W3C Recommendation, "Decryption Transform for XML Signature", 10 December 2002. |

| 1684 | **[XML-ns]** | W3C Recommendation, "Namespaces in XML," 14 January 1999. |

| 1685 1686 | **[XMLSCHEMA]** | W3C Recommendation, "XML Schema Part 1: Structures,"2 May 2001. W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001. |

| 1687 1688 | **[XMLSIG]** | W3C Recommendation, "XML Signature Syntax and Processing," 12 February 2002. |

| 1689 1690 1691 1692 | **[X509]** | S. Santesson, et al,"Internet X.509 Public Key Infrastructure Qualified Certificates Profile," http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I |

| 1693 1694 | **[WSS-SAML]** | OASIS Working Draft 06, "Web Services Security SAML Token Profile", 21 February 2003 |

1695 1696	**[WSS-XrML]**	OASIS Working Draft 03, "Web Services Security XrML Token Profile", 30 January 2003
1697 1698 1699	**[WSS-X509]**	OASIS, "Web Services Security X.509 Certificate Token Profile", 19 January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis- 200401-wss-x509-token-profile-1.0
1700 1701	**[WSSKERBEROS]**	OASIS Working Draft 03, "Web Services Security Kerberos Profile", 30 January 2003
1702 1703 1704	**[WSSUSERNAME]**	OASIS,"Web Services Security UsernameToken Profile" 19 January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss- username-token-profile-1.0
1705 1706	**[WSS-XCBF]**	OASIS Working Draft 1.1, "Web Services Security XCBF Token Profile", 30 March 2003
1707 1708	**[XPOINTER]**	"XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation", DeRose, Maler, Daniel, 11 September 2001.

1709 Appendix A: Utility Elements and Attributes

1710 These specifications define several elements, attributes, and attribute groups which can be re-
1711 used by other specifications. This appendix provides an overview of these *utility* components. It
1712 should be noted that the detailed descriptions are provided in the specification and this appendix
1713 will reference these sections as well as calling out other aspects not documented in the
1714 specification.

1715 A.1. Identification Attribute

1716 There are many situations where elements within SOAP messages need to be referenced. For
1717 example, when signing a SOAP message, selected elements are included in the signature. XML
1718 Schema Part 2 provides several built-in data types that may be used for identifying and
1719 referencing elements, but their use requires that consumers of the SOAP message either have or
1720 are able to obtain the schemas where the identity or reference mechanisms are defined. In some
1721 circumstances, for example, intermediaries, this can be problematic and not desirable.
1722 Consequently a mechanism is required for identifying and referencing elements, based on the
1723 SOAP foundation, which does not rely upon complete schema knowledge of the context in which
1724 an element is used. This functionality can be integrated into SOAP processors so that elements
1725 can be identified and referred to without dynamic schema discovery and processing.
1726 This specification specifies a namespace-qualified global attribute for identifying an element
1727 which can be applied to any element that either allows arbitrary attributes or specifically allows
1728 this attribute. This is a general purpose mechanism which can be re-used as needed.
1729 A detailed description can be found in Section 4.0 ID References.
1730
1731 This section is non-normative.

1732 A.2. Timestamp Elements

1733 The specification defines XML elements which may be used to express timestamp information
1734 such as creation and expiration. While defined in the context of message security, these
1735 elements can be re-used wherever these sorts of time statements need to be made.
1736 The elements in this specification are defined and illustrated using time references in terms of the
1737 *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this
1738 type for interoperability. It is further RECOMMENDED that all references be in UTC time for
1739 increased interoperability. If, however, other time types are used, then the ValueType attribute
1740 MUST be specified to indicate the data type of the time format.
1741 The following table provides an overview of these elements:
1742

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

1743
1744 A detailed description can be found in Section 10.
1745
1746 This section is non-normative.
1747

1748 A.3. General Schema Types

1749 The schema for the utility aspects of this specification also defines some general purpose
1750 schema elements. While these elements are defined in this schema for use with this
1751 specification, they are general purpose definitions that may be used by other specifications as
1752 well.
1753 Specifically, the following schema elements are defined and can be re-used:
1754

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

1755
1756 This section is non-normative.
1757

1758 Appendix B: SecurityTokenReference Model

1759 This appendix provides a non-normative overview of the usage and processing models for the
1760 `<wsse:SecurityTokenReference>` element.
1761 There are several motivations for introducing the `<wsse:SecurityTokenReference>`
1762 element:
1763 The XML Signature reference mechanisms are focused on "key" references rather than general
1764 token references.
1765 The XML Signature reference mechanisms utilize a fairly closed schema which limits the
1766 extensibility that can be applied.
1767 There are additional types of general reference mechanisms that are needed, but are not covered
1768 by XML Signature.
1769 There are scenarios where a reference may occur outside of an XML Signature and the XML
1770 Signature schema is not appropriate or desired.
1771 The XML Signature references may include aspects (e.g. transforms) that may not apply to all
1772 references.
1773
1774 The following use cases drive the above motivations:
1775 **Local Reference** – A security token, that is included in the message in the `<wsse:Security>`
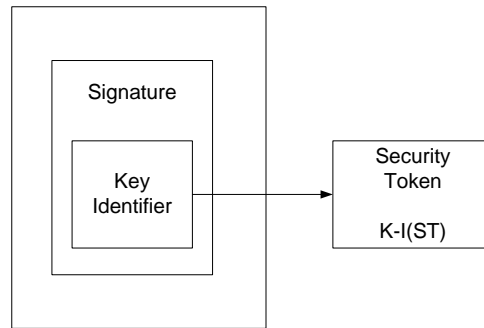1776 header, is associated with an XML Signature. The figure below illustrates this:
1777
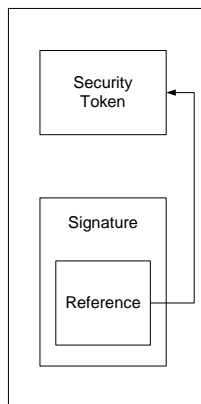
1778
1779 **Remote Reference** – A security token, that is not included in the message but may be available
1780 at a specific URI, is associated with an XML Signature. The figure below illustrates this:
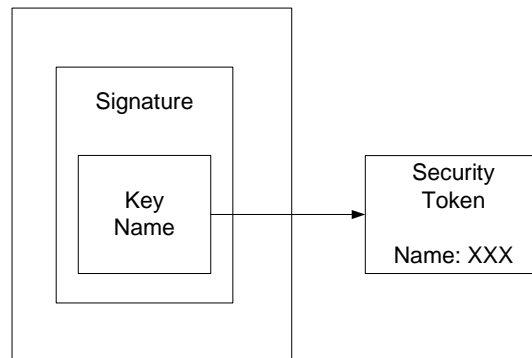
Signature
Reference → Security Token
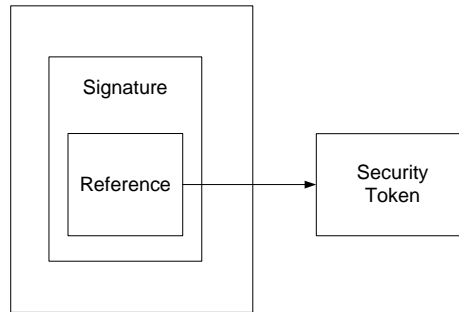
1781
1782 **Key Identifier** – A security token, which is associated with an XML Signature and identified using
1783 a known value that is the result of a well-known function of the security token (defined by the
1784 token format or profile). The figure below illustrates this where the token is located externally:

Signature
Key Identifier → Security Token K-I(ST)

1785
1786 **Key Name** – A security token is associated with an XML Signature and identified using a known
1787 value that represents a "name" assertion within the security token (defined by the token format or
1788 profile). The figure below illustrates this where the token is located externally:

Signature
Key Name → Security Token Name: XXX

1789
1790 **Format-Specific References** – A security token is associated with an XML Signature and
1791 identified using a mechanism specific to the token (rather than the general mechanisms
1792 described above). The figure below illustrates this:
1793
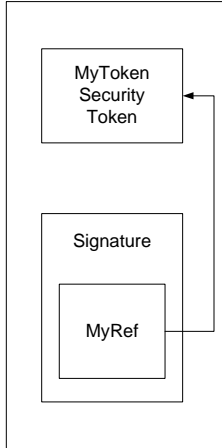
1794 **Non-Signature References** – A message may contain XML that does not represent an XML
1795 signature, but may reference a security token (which may or may not be included in the
1796 message). The figure below illustrates this:
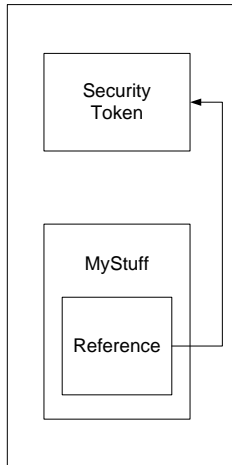
1797
1798
1799 All conformant implementations MUST be able to process the
1800 `<wsse:SecurityTokenReference>` element. However, they are not required to support all of
1801 the different types of references.
1802 The reference MAY include a *ValueType* attribute which provides a "hint" for the type of desired
1803 token.
1804 If multiple sub-elements are specified, together they describe the reference for the token.
1805 There are several challenges that implementations face when trying to interoperate:
1806 **ID References** – The underlying XML referencing mechanism using the XML base type of ID
1807 provides a simple straightforward XML element reference. However, because this is an XML
1808 type, it can be bound to *any* attribute. Consequently in order to process the IDs and references
1809 requires the recipient to *understand* the schema. This may be an expensive task and in the
1810 general case impossible as there is no way to know the "schema location" for a specific
1811 namespace URI.
1812 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID
1813 references are, by definition, unique by XML. However, other mechanisms such as "principal
1814 name" are not required to be unique and therefore such references may be unique.
1815 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide
1816 information about the "key" used in the signature. For token references within signatures, it is
1817 RECOMMENDED that the `<wsse:SecurityTokenReference>` be placed within the
1818 `<ds:KeyInfo>`. The XML Signature specification also defines mechanisms for referencing keys
1819 by identifier or passing specific keys. As a rule, the specific mechanisms defined in WSS: SOAP
1820 Message Security or its profiles are preferred over the mechanisms in XML Signature.
1821 The following provides additional details on the specific reference mechanisms defined in WSS:
1822 SOAP Message Security:
1823 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to
1824 the security token. If only the fragment is specified, then it references the security token within

1825 the document whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to
1826 a [potentially external] security token identified using a URI. There are no implied semantics
1827 around the processing of the URI.
1828 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token
1829 by specifying a known value (identifier) for the token, which is determined by applying a special
1830 *function* to the security token (e.g. a hash of key fields). This approach is typically unique for the
1831 specific security token but requires a profile or token-specific function to be specified. The
1832 `ValueType` attribute defines the type of key identifier and, consequently, identifies the type of
1833 token referenced. The `EncodingType` attribute specifies how the unique value (identifier) is
1834 encoded. For example, a hash value may be encoded using base 64 encoding (the default).
1835 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a
1836 specific value that is used to *match* an identity assertion within the security token. This is a
1837 subset match and may result in multiple security tokens that match the specified name. While
1838 XML Signature doesn't imply formatting semantics, WSS: SOAP Message Security
1839 RECOMMENDS that X.509 names be specified.
1840 It is expected that, where appropriate, profiles define if and how the reference mechanisms map
1841 to the specific token profile. Specifically, the profile should answer the following questions:
1842 • What types of references can be used?
1843 • How "Key Name" references map (if at all)?
1844 • How "Key Identifier" references map (if at all)?
1845 • Are there any additional profile or format-specific references?
1846
1847 This section is non-normative.

1848 # Appendix C: Revision History

Rev	Date	What
01	20-Sep-02	Initial draft based on input documents and editorial review
02	24-Oct-02	Update with initial comments (technical and grammatical)
03	03-Nov-02	Feedback updates
04	17-Nov-02	Feedback updates
05	02-Dec-02	Feedback updates
06	08-Dec-02	Feedback updates
07	11-Dec-02	Updates from F2F
08	12-Dec-02	Updates from F2F
14	03-Jun-03	Completed these pending issues - 62, 69, 70, 72, 74, 84, 90, 94, 95, 96, 97, 98, 99, 101, 102, 103, 106, 107, 108, 110, 111
15	18-Jul-03	Completed these pending issues – 78, 82, 104, 105, 109, 111, 113
16	26-Aug-03	Completed these pending issues - `99, 128, 130, 132, 134`
18	15-Dec-03	Editorial Updates based on Issue List #30
19	29-Dec-03	Editorial Updates based on Issue List #31
20	14-Jan-04	Completed issue 241 and feedback updates
21	19-Jan-04	Editorial corrections for name space and document name
22	17-Feb-04	Editorial changes per Karl Best

1849
1850 This section is non-normative.

1851 **Appendix D: Notices**

1852 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1853 that might be claimed to pertain to the implementation or use of the technology described in this
1854 document or the extent to which any license under such rights might or might not be available;
1855 neither does it represent that it has made any effort to identify any such rights. Information on
1856 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1857 website. Copies of claims of rights made available for publication and any assurances of licenses
1858 to be made available, or the result of an attempt made to obtain a general license or permission
1859 for the use of such proprietary rights by implementers or users of this specification, can be
1860 obtained from the OASIS Executive Director.
1861 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1862 applications, or other proprietary rights which may cover technology that may be required to
1863 implement this specification. Please address the information to the OASIS Executive Director.
1864 Copyright © OASIS Open 2002-2004. *All Rights Reserved.*
1865 This document and translations of it may be copied and furnished to others, and derivative works
1866 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1867 published and distributed, in whole or in part, without restriction of any kind, provided that the
1868 above copyright notice and this paragraph are included on all such copies and derivative works.
1869 However, this document itself does not be modified in any way, such as by removing the
1870 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1871 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1872 Property Rights document must be followed, or as required to translate it into languages other
1873 than English.
1874 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1875 successors or assigns.
1876 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1877 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1878 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1879 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1880 PARTICULAR PURPOSE.
1881
1882 This section is non-normative.