



# Web Services Security: SOAP Message Security Version 1.1.1

## Committee Specification Draft 01

23 March 2011

### Specification URIs:

#### This Version:

<http://docs.oasis-open.org/wss-m/wss/v1.1.1/csd01/wss-SOAPMessageSecurity-v1.1.1-csd01.doc> (Authoritative)  
<http://docs.oasis-open.org/wss-m/wss/v1.1.1/csd01/wss-SOAPMessageSecurity-v1.1.1-csd01.pdf>  
<http://docs.oasis-open.org/wss-m/wss/v1.1.1/csd01/wss-SOAPMessageSecurity-v1.1.1-csd01.html>

#### Previous Version:

<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SOAPMessageSecurity.doc>  
<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SOAPMessageSecurity.pdf>  
<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SOAPMessageSecurity.html>

#### Latest Version:

<http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SOAPMessageSecurity-v1.1.1.doc> (Authoritative)  
<http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SOAPMessageSecurity-v1.1.1.pdf>  
<http://docs.oasis-open.org/wss-m/wss/v1.1.1/wss-SOAPMessageSecurity-v1.1.1.html>

### Technical Committee:

OASIS Web Services Security Maintenance (WSS-M) TC

### Chair(s):

David Turner, Microsoft

### Editor(s):

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

### Related work:

This specification is one part of a multi-part Work Product. The other parts include:

[Web Services Security Kerberos Token Profile Version 1.1.1](#)  
[Web Services Security Rights Expression Language \(REL\) Token Profile Version 1.1.1](#)  
[Web Services Security SOAP Messages with Attachments \(SwA\) Profile Version 1.1.1](#)  
[Web Services Security Username Token Profile Version 1.1.1](#)  
[Web Services Security X.509 Certificate Token Profile Version 1.1.1](#)  
[Web Services Security SAML Token Profile Version 1.1.1](#)  
Schemas: <http://docs.oasis-open.org/wss-m/wss/v1.1.1/csd01/xsd/>

This specification supersedes:

- [Web Services Security: SOAP Message Security 1.1 \(WS-Security 2004\)](#), OASIS Standard Incorporating Approved Errata, 01 November 2006
- [Web Services Security: SOAP Message Security 1.1 \(WS-Security 2004\)](#), OASIS Approved Errata, 01 November 2006

#### Abstract:

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

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

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

#### Status:

This document was last revised or approved by the [OASIS Web Services Security Maintenance \(WSS-M\) TC](#) on the above date. The level of approval is also listed above. Check the “Latest Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “[Send A Comment](#)” button on the Technical Committee’s web page at <http://www.oasis-open.org/committees/wss-m/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/wss-m/ipr.php>).

This document integrates specific error corrections or editorial changes to the preceding specification, within the scope of the Web Services Security and this TC.

This document introduces a third digit in the numbering convention where the third digit represents a consolidation of error corrections, bug fixes or editorial formatting changes (e.g., 1.1.1); it does not add any new features beyond those of the base specifications (e.g., 1.1).

#### Citation Format:

##### **[WSS-SOAP-Message-Security-V1.1.1]**

*Web Services Security: SOAP Message Security Version 1.1.1*. 23 March 2011. OASIS Committee Specification Draft 01. <http://docs.oasis-open.org/wss-m/wss/v1.1.1/csd01/wss-SOAPMessageSecurity-v1.1.1-csd01.doc>.

---

## Notices

Copyright © OASIS Open 2011. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

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

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

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

---

# Table of Contents

1	Introduction .....	6
1.1	Goals and Requirements .....	6
1.1.1	Requirements .....	6
1.1.2	Non-Goals .....	7
2	Notations and Terminology .....	8
2.1	Notational Conventions.....	8
2.2	Namespaces.....	8
2.3	Acronyms and Abbreviations .....	9
2.4	Terminology .....	9
2.5	Note on Examples .....	11
3	Message Protection Mechanisms .....	12
3.1	Message Security Model .....	12
3.2	Message Protection .....	12
3.3	Invalid or Missing Claims .....	13
3.4	Example .....	13
4	ID References.....	15
4.1	Id Attribute.....	15
4.2	Id Schema.....	16
5	Security Header.....	17
6	Security Tokens.....	19
6.1	Attaching Security Tokens .....	19
6.1.1	Processing Rules .....	19
6.1.2	Subject Confirmation .....	19
6.2	User Name Token.....	19
6.2.1	Usernames .....	19
6.3	Binary Security Tokens.....	20
6.3.1	Attaching Security Tokens.....	20
6.3.2	Encoding Binary Security Tokens .....	20
6.4	XML Tokens.....	21
6.5	EncryptedData Token .....	21
6.6	Identifying and Referencing Security Tokens .....	22
7	Token References .....	23
7.1	SecurityTokenReference Element.....	23
7.2	Direct References .....	25
7.3	Key Identifiers .....	26
7.4	Embedded References .....	27
7.5	ds:KeyInfo .....	28
7.6	Key Names .....	29
7.7	Encrypted Key reference .....	29
8	Signatures .....	30
8.1	Algorithms .....	30
8.2	Signing Messages .....	32

8.3	Signing Tokens .....	33
8.4	Signature Validation.....	36
8.5	Signature Confirmation .....	36
8.5.1	Response Generation Rules .....	37
8.5.2	Response Processing Rules .....	37
8.6	Example .....	38
9	Encryption.....	39
9.1	xenc:ReferenceList .....	39
9.2	xenc:EncryptedKey .....	40
9.3	Encrypted Header .....	41
9.4	Processing Rules .....	41
9.4.1	Encryption.....	41
9.4.2	Decryption .....	42
9.4.3	Encryption with EncryptedHeader .....	42
9.4.4	Processing an EncryptedHeader.....	43
9.4.5	Processing the mustUnderstand attribute on EncryptedHeader .....	43
10	Security Timestamps .....	44
11	Extended Example .....	46
12	Error Handling .....	49
13	Security Considerations .....	51
13.1	General Considerations .....	51
13.2	Additional Considerations .....	51
13.2.1	Replay .....	51
13.2.2	Combining Security Mechanisms .....	52
13.2.3	Challenges.....	52
13.2.4	Protecting Security Tokens and Keys .....	52
13.2.5	Protecting Timestamps and Ids.....	53
13.2.6	Protecting against removal and modification of XML Elements.....	53
13.2.7	Detecting Duplicate Identifiers.....	54
14	Interoperability Notes.....	55
15	Privacy Considerations.....	56
16	References .....	57
#	Conformance.....	59
A.	Acknowledgements .....	60
B.	Revision History.....	64
C.	Utility Elements and Attributes .....	65
C.1	Identification Attribute.....	65
C.2	Timestamp Elements .....	65
C.3	General Schema Types .....	66
D.	SecurityTokenReference Model .....	67

---

# 1 Introduction

This OASIS specification is the result of significant new work by the WSS Technical Committee and supersedes the input submissions, Web Service Security (WS-Security) Version 1.0 April 5, 2002 and Web Services Security Addendum Version 1.0 August 18, 2002.

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

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

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

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

## 1.1 Goals and Requirements

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

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

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

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

The requirements to support secure message exchange are listed below.

### 1.1.1 Requirements

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

- 44 • Multiple security token formats
- 45 • Multiple trust domains
- 46 • Multiple signature formats
- 47 • Multiple encryption technologies
- 48 • End-to-end message content security and not just transport-level security

### 49 **1.1.2 Non-Goals**

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

- 51 • Establishing a security context or authentication mechanisms.
- 52 • Key derivation.
- 53 • Advertisement and exchange of security policy.
- 54 • How trust is established or determined.
- 55 • Non-repudiation.

56

---

## 2 Notations and Terminology

57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97

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

### 2.1 Notational Conventions

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

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

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

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

### 2.2 Namespaces

Namespace URIs (of the general form "some-URI") represents some application-dependent or context-dependent URI as defined in RFC 2396 [URI].

This specification is backwardly compatible with version 1.0. This means that URIs and schema elements defined in 1.0 remain unchanged and new schema elements and constants are defined using 1.1 namespaces and URIs.

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

```
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd
```

This specification is designed to work with the general SOAP [SOAP11, SOAP12] message structure and message processing model, and should be applicable to any version of SOAP. The current SOAP 1.1 namespace URI is used herein to provide detailed examples, but there is no intention to limit the applicability of this specification to a single version of SOAP.

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

Prefix	Namespace
ds	<a href="http://www.w3.org/2000/09/xmldsig#">http://www.w3.org/2000/09/xmldsig#</a>
s11	<a href="http://schemas.xmlsoap.org/soap/envelope/">http://schemas.xmlsoap.org/soap/envelope/</a>
s12	<a href="http://www.w3.org/2003/05/soap-envelope">http://www.w3.org/2003/05/soap-envelope</a>
wsse	<a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd</a>
wsse11	<a href="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd">http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd</a>
wsu	<a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd</a>
xenc	<a href="http://www.w3.org/2001/04/xmlenc#">http://www.w3.org/2001/04/xmlenc#</a>

98

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

100

101 URI fragments defined in this document are relative to the following base URI unless otherwise stated:

102 <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0>

## 103 2.3 Acronyms and Abbreviations

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

105

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

## 106 2.4 Terminology

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

108

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

111  
112 **Claim Confirmation** – A *claim confirmation* is the process of verifying that a claim applies to an entity.

113  
114 **Confidentiality** – *Confidentiality* is the property that data is not made available to unauthorized  
115 individuals, entities, or processes.

116  
117 **Digest** – A *digest* is a cryptographic checksum of an octet stream.

118  
119 **Digital Signature** – A *digital signature* is a value computed with a cryptographic algorithm and bound  
120 to data in such a way that intended recipients of the data can use the digital signature to verify that the  
121 data has not been altered and/or has originated from the signer of the message, providing message  
122 integrity and authentication. The digital signature can be computed and verified with symmetric key  
123 algorithms, where the same key is used for signing and verifying, or with asymmetric key algorithms,  
124 where different keys are used for signing and verifying (a private and public key pair are used).

125  
126 **End-To-End Message Level Security** – *End-to-end message level security* is established when  
127 a message that traverses multiple applications (one or more SOAP intermediaries) within and between  
128 business entities, e.g. companies, divisions and business units, is secure over its full route through and  
129 between those business entities. This includes not only messages that are initiated within the entity but  
130 also those messages that originate outside the entity, whether they are Web Services or the more  
131 traditional messages.

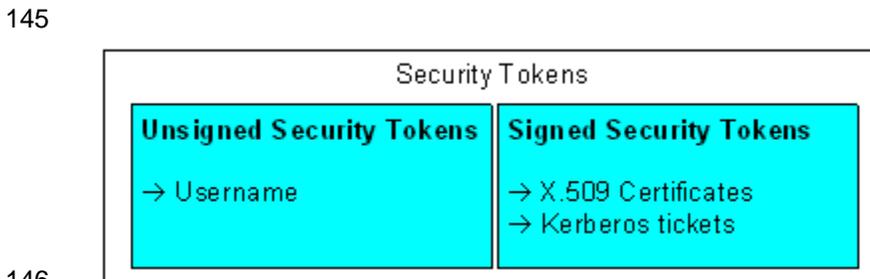
132  
133 **Integrity** – *Integrity* is the property that data has not been modified.

134  
135 **Message Confidentiality** - *Message Confidentiality* is a property of the message and encryption is  
136 the mechanism by which this property of the message is provided.

137  
138 **Message Integrity** - *Message Integrity* is a property of the message and digital signature is a  
139 mechanism by which this property of the message is provided.

140  
141 **Signature** - In this document, signature and digital signature are used interchangeably and have the  
142 same meaning.

143  
144 **Security Token** – A *security token* represents a collection (one or more) of claims.



146  
147  
148 **Signed Security Token** – A *signed security token* is a security token that is asserted and  
149 cryptographically signed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket).

150

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

## 153 **2.5 Note on Examples**

154 The examples which appear in this document are only intended to illustrate the correct syntax of the  
155 features being specified. The examples are NOT intended to necessarily represent best practice for  
156 implementing any particular security properties.

157

158 Specifically, the examples are constrained to contain only mechanisms defined in this document. The  
159 only reason for this is to avoid requiring the reader to consult other documents merely to understand the  
160 examples. It is NOT intended to suggest that the mechanisms illustrated represent best practice or are  
161 the strongest available to implement the security properties in question. In particular, mechanisms defined  
162 in other Token Profiles are known to be stronger, more efficient and/or generally superior to some of the  
163 mechanisms shown in the examples in this document.

---

## 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 attacker or
- an antagonist could send messages to a service that, while well-formed, lack appropriate security claims to warrant processing
- an antagonist could alter a message to the service which being well formed causes the service to process and respond to the client for an incorrect request.

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.

207

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

211

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

### 214 3.3 Invalid or Missing Claims

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

### 221 3.4 Example

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

228

```
229 (001) <?xml version="1.0" encoding="utf-8"?>
230 (002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
231 (003)   xmlns:ds="...">
232 (004)   <S11:Header>
233 (005)     <wsse:Security
234 (006)       xmlns:wsse="...">
235 (007)       <wsse:BinarySecurityToken ValueType=" http://fabrikam123#CustomToken
236 (008) "
237 (009)       EncodingType="...#Base64Binary" wsu:Id=" MyID ">
238 (010)         FHUIORv...
239 (011)       </wsse:BinarySecurityToken>
240 (012)       <ds:Signature>
241 (013)         <ds:SignedInfo>
242 (014)           <ds:CanonicalizationMethod
243 (015)             Algorithm=
244 (016)               "http://www.w3.org/2001/10/xml-exc-c14n#" />
245 (017)           <ds:SignatureMethod
246 (018)             Algorithm=
247 (019)               "http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
248 (020)           <ds:Reference URI="#MsgBody">
249 (021)             <ds:DigestMethod
250 (022)               Algorithm=
251 (023)                 "http://www.w3.org/2000/09/xmldsig#sha1" />
252 (024)             <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
253 (025)           </ds:Reference>
254 (026)         </ds:SignedInfo>
255 (027)         <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
256 (028)         <ds:KeyInfo>
257 (029)           <wsse:SecurityTokenReference>
258 (030)             <wsse:Reference URI="#MyID" />
259 (031)           </wsse:SecurityTokenReference>
```

```

260      (022)          </ds:KeyInfo>
261      (023)          </ds:Signature>
262      (024)          </wsse:Security>
263      (025)          </S11:Header>
264      (026)          <S11:Body wsu:Id="MsgBody">
265      (027)          <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
266                      QQQ
267          </tru:StockSymbol>
268      (028)          </S11:Body>
269      (029)          </S11:Envelope>

```

270

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

273

274 Line (004) starts the [<wsse:Security>](#) header defined in this specification. This header contains  
275 security information for an intended recipient. This element continues until line (024).

276

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

279

280 Lines (008) to (023) specify a digital signature. This signature ensures the [integrity](#) of the signed  
281 elements. The signature uses the [XML Signature](#) specification identified by the ds namespace  
282 declaration in Line (002).

283

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

285

286 Line (010) specifies how to canonicalize (normalize) the data that is being signed. Lines (012) to (015)  
287 select the elements that are signed and how to digest them. Specifically, line (012) indicates that the  
288 [<S11:Body>](#) element is signed. In this example only the message body is signed; typically all critical  
289 elements of the message are included in the signature (see the [Extended Example](#) below).

290

291 Line (017) specifies the signature value of the canonicalized form of the data that is being signed as  
292 defined in the [XML Signature](#) specification.

293

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

297

298 Lines (026) to (028) contain the body (payload) of the [SOAP](#) message.

299

## 4 ID References

300 There are many motivations for referencing other message elements such as signature references or  
301 correlating signatures to security tokens. For this reason, this specification defines the `wsu:Id` attribute  
302 so that recipients need not understand the full schema of the message for processing of the security  
303 elements. That is, they need only "know" that the `wsu:Id` attribute represents a schema type of ID which  
304 is used to reference elements. However, because some key schemas used by this specification don't  
305 allow attribute extensibility (namely XML Signature and XML Encryption), this specification also allows  
306 use of their local ID attributes in addition to the `wsu:Id` attribute and the `xml:id` attribute [XMLID]. As a  
307 consequence, when trying to locate an element referenced in a signature, the following attributes are  
308 considered (in no particular order):

309

- 310 • Local ID attributes on XML Signature elements
- 311 • Local ID attributes on XML Encryption elements
- 312 • Global `wsu:Id` attributes (described below) on elements
- 313 • Profile specific defined identifiers
- 314 • Global `xml:id` attributes on elements

315

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

319

320 Tokens and elements that are defined in this specification and related profiles to use `wsu:Id` attributes  
321 SHOULD use `wsu:Id`. Elements to be signed MAY use `xml:id` [XMLID] or `wsu:Id`, and use of `xml:id`  
322 MAY be specified in profiles. All receivers MUST be able to identify XML elements carrying a `wsu:Id`  
323 attribute as representing an attribute of schema type ID and process it accordingly.

324

325 All receivers MAY be able to identify XML elements with a `xml:id` attribute as representing an ID  
326 attribute and process it accordingly. Senders SHOULD use `wsu:Id` and MAY use `xml:id`. Note that use  
327 of `xml:id` in conjunction with inclusive canonicalization may be inappropriate, as noted in [XMLID] and  
328 thus this combination SHOULD be avoided.

329

### 4.1 Id Attribute

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

337

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

342

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

345  
346 Alternatively, the `xml:id` attribute MAY be used. Applications MUST NOT specify both a `wsu:Id` and  
347 `xml:id` attribute on a single element. It is an XML requirement that only one id attribute be specified on a  
348 single element.

## 349 4.2 Id Schema

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

353 The syntax for this attribute is as follows:

```
354  
355 <anyElement wsu:Id="...">...</anyElement>
```

356  
357 The following describes the attribute illustrated above:

358 `.../@wsu:Id`

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

361  
362 Two `wsu:Id` attributes within an XML document MUST NOT have the same value. Implementations MAY  
363 rely on XML Schema validation to provide rudimentary enforcement for intra-document uniqueness.  
364 However, applications SHOULD NOT rely on schema validation alone to enforce uniqueness.

365  
366 This specification does not specify how this attribute will be used and it is expected that other  
367 specifications MAY add additional semantics (or restrictions) for their usage of this attribute.

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

```
369  
370 <x:myElement wsu:Id="ID1" xmlns:x="..."  
371 xmlns:wsu="...">
```

372  
373 Conformant processors that do support XML Schema MUST treat this attribute as if it was defined using a  
374 global attribute declaration.

375  
376 Conformant processors that do not support dynamic XML Schema or DTDs discovery and processing are  
377 strongly encouraged to integrate this attribute definition into their parsers. That is, to treat this attribute  
378 information item as if its PSVI has a [type definition] which {target namespace} is  
379 "`http://www.w3.org/2001/XMLSchema`" and which {type} is "ID." Doing so allows the processor to  
380 inherently know *how* to process the attribute without having to locate and process the associated schema.  
381 Specifically, implementations MAY support the value of the `wsu:Id` as the valid identifier for use as an  
382 [XPointer](#) [XPointer] shorthand pointer for interoperability with XML Signature references.

383

---

## 384 5 Security Header

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

391  
392 As stated, a message MAY have multiple `<wsse:Security>` header blocks if they are targeted for  
393 separate recipients. A message MUST NOT have multiple `<wsse:Security>` header blocks targeted  
394 (whether explicitly or implicitly) at the same recipient. However, only one `<wsse:Security>` header  
395 block MAY omit the `S11:actor` or `S12:role` attributes. Two `<wsse:Security>` header blocks MUST  
396 NOT have the same value for `S11:actor` or `S12:role`. Message security information targeted for  
397 different recipients MUST appear in different `<wsse:Security>` header blocks. This is due to potential  
398 processing order issues (e.g. due to possible header re-ordering). The `<wsse:Security>` header block  
399 without a specified `S11:actor` or `S12:role` MAY be processed by anyone, but MUST NOT be removed  
400 prior to the final destination or endpoint.

401  
402 As elements are added to a `<wsse:Security>` header block, they SHOULD be prepended to the  
403 existing elements. As such, the `<wsse:Security>` header block represents the signing and encryption  
404 steps the message producer took to create the message. This prepending rule ensures that the receiving  
405 application can process sub-elements in the order they appear in the `<wsse:Security>` header block,  
406 because there will be no forward dependency among the sub-elements. Note that this specification does  
407 not impose any specific order of processing the sub-elements. The receiving application can use  
408 whatever order is required.

409  
410 When a sub-element refers to a key carried in another sub-element (for example, a signature sub-  
411 element that refers to a binary security token sub-element that contains the [X.509](#) certificate used for the  
412 signature), the key-bearing element SHOULD be ordered to precede the key-using

413 Element:

```
414  
415 <S11:Envelope>  
416   <S11:Header>  
417     ...  
418     <wsse:Security S11:actor="..." S11:mustUnderstand="...">  
419       ...  
420     </wsse:Security>  
421     ...  
422   </S11:Header>  
423   ...  
424 </S11:Envelope>
```

425  
426 The following describes the attributes and elements listed in the example above:

427 `/wsse:Security`

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

429  
430 `/wsse:Security/@S11:actor`

431 This attribute allows a specific SOAP 1.1 [SOAP11] actor to be identified. This attribute is  
432 optional; however, no two instances of the header block may omit an actor or specify the same  
433 actor.

434 */wsse:Security/@S12:role*

436 This attribute allows a specific SOAP 1.2 [SOAP12] role to be identified. This attribute is optional;  
437 however, no two instances of the header block may omit a role or specify the same role.

438  
439 */wsse:Security/@S11:mustUnderstand*

440 This SOAP 1.1 [SOAP11] attribute is used to indicate whether a header entry is mandatory or  
441 optional for the recipient to process. The value of the mustUnderstand attribute is either "1" or "0".  
442 The absence of the SOAP mustUnderstand attribute is semantically equivalent to its presence  
443 with the value "0".

444  
445 */wsse:Security/@S12:mustUnderstand*

446 This SOAP 1.2 [SOAP12] attribute is used to indicate whether a header entry is mandatory or  
447 optional for the recipient to process. The value of the mustUnderstand attribute is either "true", "1",  
448 "false" or "0". The absence of the SOAP mustUnderstand attribute is semantically equivalent to  
449 its presence with the value "false".

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

452 This is an extensibility mechanism to allow different (extensible) types of security information,  
453 based on a schema, to be passed. Unrecognized elements SHOULD cause a fault.

454  
455 */wsse:Security/@{any}*

456 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added  
457 to the header. Unrecognized attributes SHOULD cause a fault.

458  
459 All compliant implementations MUST be able to process a `<wsse:Security>` element.

460  
461 All compliant implementations MUST declare which profiles they support and MUST be able to process a  
462 `<wsse:Security>` element including any sub-elements which may be defined by that profile. It is  
463 RECOMMENDED that undefined elements within the `<wsse:Security>` header not be processed.

464  
465 The next few sections outline elements that are expected to be used within a `<wsse:Security>`  
466 header.

467  
468 When a `<wsse:Security>` header includes a `mustUnderstand="true"` attribute:

- 469 • The receiver MUST generate a SOAP fault if does not implement the WSS: SOAP Message  
470 Security specification corresponding to the namespace. Implementation means ability to interpret  
471 the schema as well as follow the required processing rules specified in WSS: SOAP Message  
472 Security.
- 473 • The receiver MUST generate a fault if unable to interpret or process security tokens contained in  
474 the `<wsse:Security>` header block according to the corresponding WSS: SOAP Message  
475 Security token profiles.
- 476 • Receivers MAY ignore elements or extensions within the `<wsse:Security>` element, based on  
477 local security policy.

---

## 478 6 Security Tokens

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

### 480 6.1 Attaching Security Tokens

481 This specification defines the `<wsse:Security>` header as a mechanism for conveying security  
482 information with and about a [SOAP](#) message. This header is, by design, extensible to support many  
483 types of security information.

484

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

#### 487 6.1.1 Processing Rules

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

#### 492 6.1.2 Subject Confirmation

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

## 496 6.2 User Name Token

### 497 6.2.1 Usernames

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

500 The following illustrates the syntax of this element:

501

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

505

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

507

508 */wsse:UsernameToken*

509 This element is used to represent a claimed identity.

510

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

512 A string label for this security token. The `wsu:Id` allow for an open attribute model.

513

514 */wsse:UsernameToken/wsse:Username*

515 This required element specifies the claimed identity.

516

517 `/wsse:UsernameToken/wsse:Username/@{any}`  
518 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added  
519 to the `<wsse:Username>` element.

520  
521 `/wsse:UsernameToken/{any}`  
522 This is an extensibility mechanism to allow different (extensible) types of security information,  
523 based on a schema, to be passed. Unrecognized elements SHOULD cause a fault.

524  
525 `/wsse:UsernameToken/@{any}`  
526 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added  
527 to the `<wsse:UsernameToken>` element. Unrecognized attributes SHOULD cause a fault.

528  
529 All compliant implementations MUST be able to process a `<wsse:UsernameToken>` element.

530 The following illustrates the use of this:

531

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

## 545 6.3 Binary Security Tokens

### 546 6.3.1 Attaching Security Tokens

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

### 549 6.3.2 Encoding Binary Security Tokens

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

554

555 The `<wsse:BinarySecurityToken>` element defines two attributes that are used to interpret it. The  
556 `ValueType` attribute indicates what the security token is, for example, a [Kerberos](#) ticket.

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

558 The following is an overview of the syntax:

559

```
560 <wsse:BinarySecurityToken wsu:Id=...  
561   EncodingType=...  
562   ValueType=.../>
```

563

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

565 */wsse:BinarySecurityToken*  
 566 This element is used to include a binary-encoded security token.  
 567  
 568 */wsse:BinarySecurityToken/@wsu:Id*  
 569 An optional string label for this [security token](#).  
 570  
 571 */wsse:BinarySecurityToken/@ValueType*  
 572 The `ValueType` attribute is used to indicate the "value space" of the encoded binary data (e.g.  
 573 an [X.509](#) certificate). The `ValueType` attribute allows a URI that defines the value type and  
 574 space of the encoded binary data. Subsequent specifications **MUST** define the `ValueType` value  
 575 for the tokens that they define. The usage of `ValueType` is **RECOMMENDED**.  
 576  
 577 */wsse:BinarySecurityToken/@EncodingType*  
 578 The `EncodingType` attribute is used to indicate, using a URI, the encoding format of the binary  
 579 data (e.g., `base64` encoded). A new attribute is introduced, as there are issues with the current  
 580 schema validation tools that make derivations of mixed simple and complex types difficult within  
 581 [XML Schema](#). The `EncodingType` attribute is interpreted to indicate the encoding format of the  
 582 element. The following encoding formats are pre-defined:  
 583

URI	Description
#Base64Binary (default)	<a href="#">XML Schema</a> base 64 encoding

584  
 585 */wsse:BinarySecurityToken/@{any}*  
 586 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added.  
 587  
 588 All compliant implementations **MUST** be able to process a `<wsse:BinarySecurityToken>` element.

## 589 6.4 XML Tokens

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

## 592 6.5 EncryptedData Token

593 In certain cases it is desirable that the token included in the `<wsse:Security>` header be encrypted for  
 594 the recipient processing role. In such a case the `<xenc:EncryptedData>` element **MAY** be used to  
 595 contain a security token and included in the `<wsse:Security>` header. That is this specification  
 596 defines the usage of `<xenc:EncryptedData>` to encrypt security tokens contained in  
 597 `<wsse:Security>` header.

598  
 599 It should be noted that token references are not made to the `<xenc:EncryptedData>` element, but  
 600 instead to the token represented by the clear-text, once the `<xenc:EncryptedData>` element has been  
 601 processed (decrypted). Such references utilize the token profile for the contained token. i.e.,  
 602 `<xenc:EncryptedData>` **SHOULD NOT** include an XML ID for referencing the contained security  
 603 token.

604  
 605 All `<xenc:EncryptedData>` tokens **SHOULD** either have an embedded encryption key or should be  
 606 referenced by a separate encryption key.

607 When a `<xenc:EncryptedData>` token is processed, it is replaced in the message infoset with its  
608 decrypted form.

## 609 **6.6 Identifying and Referencing Security Tokens**

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

---

## 614 7 Token References

615 This chapter discusses and defines mechanisms for referencing security tokens and other key bearing  
616 elements..

### 617 7.1 SecurityTokenReference Element

618 Digital signature and encryption operations require that a key be specified. For various reasons, the  
619 element containing the key in question may be located elsewhere in the message or completely outside  
620 the message. The `<wsse:SecurityTokenReference>` element provides an extensible mechanism for  
621 referencing security tokens and other key bearing elements.

622  
623 The `<wsse:SecurityTokenReference>` element provides an open content model for referencing key  
624 bearing elements because not all of them support a common reference pattern. Similarly, some have  
625 closed schemas and define their own reference mechanisms. The open content model allows appropriate  
626 reference mechanisms to be used.

627  
628 If a `<wsse:SecurityTokenReference>` is used outside of the security header processing block the  
629 meaning of the response and/or processing rules of the resulting references MUST be specified by the  
630 the specific profile and are out of scope of this specification.

631 The following illustrates the syntax of this element:

632

```
633 <wsse:SecurityTokenReference wsu:Id="...", wsse11:TokenType="...",  
634 wsse:Usage="...", wsse:Usage="...">  
635 </wsse:SecurityTokenReference>
```

636

637 The following describes the elements defined above:

638

639 */wsse:SecurityTokenReference*

640 This element provides a reference to a security token.

641

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

643 A string label for this [security token](#) reference which names the reference. This attribute does not  
644 indicate the ID of what is being referenced, that SHOULD be done using a fragment URI in a  
645 `<wsse:Reference>` element within the `<wsse:SecurityTokenReference>` element.

646

647 */wsse:SecurityTokenReference/@wsse11:TokenType*

648 This optional attribute is used to identify, by URI, the type of the referenced token.

649 This specification recommends that token specific profiles define appropriate token type  
650 identifying URI values, and that these same profiles require that these values be specified in the  
651 profile defined reference forms.

652

653 When a `wsse11:TokenType` attribute is specified in conjunction with a

654 `wsse:KeyIdentifier/@ValueType` attribute or a `wsse:Reference/@ValueType`

655 attribute that indicates the type of the referenced token, the security token type identified by the

656 `wsse11:TokenType` attribute MUST be consistent with the security token type identified by the

657 `wsse:ValueType` attribute.

URI	Description
<pre>http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey</pre>	<p>A token type of an <code>&lt;xenc:EncryptedKey&gt;</code></p>

659

660 */wsse:SecurityTokenReference/@wsse:Usage*661 This optional attribute is used to type the usage of the `<wsse:SecurityTokenReference>`.

662 Usages are specified using URIs and multiple usages MAY be specified using XML list

663 semantics. No usages are defined by this specification.

664

665 */wsse:SecurityTokenReference/{any}*666 This is an extensibility mechanism to allow different (extensible) types of security references,  
667 based on a schema, to be passed. Unrecognized elements SHOULD cause a fault.

668

669 */wsse:SecurityTokenReference/@{any}*670 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added  
671 to the header. Unrecognized attributes SHOULD cause a fault.

672

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

675

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

680

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

687

688 This specification allows for the use of multiple reference mechanisms within a single  
689 `<wsse:SecurityTokenReference>`. When multiple references are present in a given  
690 `<wsse:SecurityTokenReference>`, they MUST resolve to a single token in common. Specific token  
691 profiles SHOULD define the reference mechanisms to be used.

692

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

695

- 696 • **Direct References** – This allows references to included tokens using URI fragments and external  
697 tokens using full URIs.

- 698 • **Key Identifiers** – This allows tokens to be referenced using an opaque value that represents the  
699 token (defined by token type/profile).
- 700 • **Key Names** – This allows tokens to be referenced using a string that matches an identity  
701 assertion within the security token. This is a subset match and may result in multiple security  
702 tokens that match the specified name.
- 703 • **Embedded References** - This allows tokens to be embedded (as opposed to a pointer to a  
704 token that resides elsewhere).

## 705 7.2 Direct References

706 The `<wsse:Reference>` element provides an extensible mechanism for directly referencing [security](#)  
707 [tokens](#) using URIs.

708

709 The following illustrates the syntax of this element:

710

```
711 <wsse:SecurityTokenReference wsu:Id="...">  
712   <wsse:Reference URI="..." ValueType="..." />  
713 </wsse:SecurityTokenReference>
```

714

715 The following describes the elements defined above:

716

717 */wsse:SecurityTokenReference/wsse:Reference*

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

719

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

721 This optional attribute specifies an abstract URI for a security token. If a fragment is specified,  
722 then it indicates the local ID of the security token being referenced. The URI **MUST** identify a  
723 security token. The URI **MUST NOT** identify a `<wsse:SecurityTokenReference>` element,  
724 a `<wsse:Embedded>` element, a `<wsse:Reference>` element, or a `<wsse:KeyIdentifier>`  
725 element.

726

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

728 This optional attribute specifies a URI that is used to identify the *type* of token being referenced.  
729 This specification does not define any processing rules around the usage of this attribute,  
730 however, specifications for individual token types **MAY** define specific processing rules and  
731 semantics around the value of the URI and its interpretation. If this attribute is not present, the  
732 URI **MUST** be processed as a normal URI.

733

734 In this version of the specification the use of this attribute to identify the type of the referenced  
735 security token is deprecated. Profiles which require or recommend the use of this attribute to  
736 identify the type of the referenced security token **SHOULD** evolve to require or recommend the  
737 use of the `wsse:SecurityTokenReference/@wsse11:TokenType` attribute to identify the  
738 type of the referenced token.

739

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

741 This is an extensibility mechanism to allow different (extensible) types of security references,  
742 based on a schema, to be passed. Unrecognized elements **SHOULD** cause a fault.

743

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

745 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added  
746 to the header. Unrecognized attributes **SHOULD** cause a fault.

747

748 The following illustrates the use of this element:

749

```
750 <wsse:SecurityTokenReference
751     xmlns:wsse="...">
752     <wsse:Reference
753         URI="http://www.fabrikam123.com/tokens/Zoe"/>
754 </wsse:SecurityTokenReference>
```

## 755 7.3 Key Identifiers

756 Alternatively, if a direct reference is not used, then it is RECOMMENDED that a key identifier be used to  
757 specify/reference a security token instead of a <ds:KeyName>. A <wsse:KeyIdentifier> is a value  
758 that can be used to uniquely identify a security token (e.g. a hash of the important elements of the  
759 security token). The exact value type and generation algorithm varies by security token type (and  
760 sometimes by the data within the token), Consequently, the values and algorithms are described in the  
761 token-specific profiles rather than this specification.

762

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

765

766 The processing model assumes that the key identifier for a security token is constant. Consequently,  
767 processing a key identifier involves simply looking for a security token whose key identifier matches the  
768 specified constant. The <wsse:KeyIdentifier> element is only allowed inside a  
769 <wsse:SecurityTokenReference> element

770 The following is an overview of the syntax:

771

```
772 <wsse:SecurityTokenReference>
773     <wsse:KeyIdentifier wsu:Id="..."
774         ValueType="..."
775         EncodingType="...">
776         ...
777     </wsse:KeyIdentifier>
778 </wsse:SecurityTokenReference>
```

779

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

781

782 */wsse:SecurityTokenReference/wsse:KeyIdentifier*

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

784

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

786 An optional string label for this identifier.

787

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

789 The optional `ValueType` attribute is used to indicate the type of `KeyIdentifier` being used. This  
790 specification defines one `ValueType` that can be applied to all token types. Each specific token profile  
791 specifies the `KeyIdentifier` types that may be used to refer to tokens of that type. It also specifies the  
792 critical semantics of the identifier, such as whether the `KeyIdentifier` is unique to the key or the token.  
793 If no value is specified then the key identifier will be interpreted in an application-specific manner. This  
794 URI fragment is relative to a base URI as indicated in the table below.

795

URI	Description
http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1	If the security token type that the Security Token Reference refers to already contains a representation for the thumbprint, the value obtained from the token MAY be used. If the token does not contain a representation of a thumbprint, then the value of the <code>KeyIdentifier</code> MUST be the SHA1 of the raw octets which would be encoded within the security token element were it to be included. A thumbprint reference MUST occur in combination with a required to be supported (by the applicable profile) reference form unless a thumbprint reference is among the reference forms required to be supported by the applicable profile, or the parties to the communication have agreed to accept thumbprint only references.
http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKeySHA1	If the security token type that the Security Token Reference refers to already contains a representation for the <code>EncryptedKey</code> , the value obtained from the token MAY be used. If the token does not contain a representation of a <code>EncryptedKey</code> , then the value of the <code>KeyIdentifier</code> MUST be the SHA1 of the raw octets which would be encoded within the security token element were it to be included.

796  
797  
798  
799  
800  
801  
802  
803  
804

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

The optional `EncodingType` attribute is used to indicate, using a URI, the encoding format of the `KeyIdentifier` (`#Base64Binary`). This specification defines the `EncodingType` URI values appearing in the following table. A token specific profile MAY define additional token specific `EncodingType` URI values. A `KeyIdentifier` MUST include an `EncodingType` attribute when its `ValueType` is not sufficient to identify its encoding type. The base values defined in this specification are:

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

805  
806  
807

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

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

## 808 7.4 Embedded References

809 In some cases a reference may be to an embedded token (as opposed to a pointer to a token that resides  
810 elsewhere). To do this, the `<wsse:Embedded>` element is specified within a  
811 `<wsse:SecurityTokenReference>` element. The `<wsse:Embedded>` element is only allowed inside  
812 a `<wsse:SecurityTokenReference>` element.

813 The following is an overview of the syntax:

814

```
815 <wsse:SecurityTokenReference>
816   <wsse:Embedded wsu:Id="...">
817     ...
818   </wsse:Embedded>
819 </wsse:SecurityTokenReference>
```

820

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

822

823 */wsse:SecurityTokenReference/wsse:Embedded*

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

825

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

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

828

829 */wsse:SecurityTokenReference/wsse:Embedded/{any}*

830 This is an extensibility mechanism to allow any security token, based on schemas, to be embedded. Unrecognized elements SHOULD cause a fault.

831

832 */wsse:SecurityTokenReference/wsse:Embedded/@{any}*

833 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added. Unrecognized attributes SHOULD cause a fault.

834

835 The following example illustrates embedding a SAML assertion:

836

```
837 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
838   <S11:Header>
839     <wsse:Security>
840       ...
841       <wsse:SecurityTokenReference>
842         <wsse:Embedded wsu:Id="tok1">
843           <saml:Assertion xmlns:saml="...">
844             ...
845           </saml:Assertion>
846         </wsse:Embedded>
847       </wsse:SecurityTokenReference>
848     </wsse:Security>
849   </S11:Header>
850   ...
851 </S11:Envelope>
```

## 857 7.5 ds:KeyInfo

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

863

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

865

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

## 869 7.6 Key Names

870 It is strongly RECOMMENDED to use `<wsse:KeyIdentifier>` elements. However, if key names are  
871 used, then it is strongly RECOMMENDED that `<ds:KeyName>` elements conform to the attribute names  
872 in section 2.3 of RFC 2253 (this is recommended by XML Signature for `<ds:X509SubjectName>`) for  
873 interoperability.

874

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

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

## 877 7.7 Encrypted Key reference

878 In certain cases, an `<xenc:EncryptedKey>` element MAY be used to carry key material encrypted for  
879 the recipient's key. This key material is henceforth referred to as `EncryptedKey`.

880

881 The `EncryptedKey` MAY be used to perform other cryptographic operations within the same message,  
882 such as signatures. The `EncryptedKey` MAY also be used for performing cryptographic operations in  
883 subsequent messages exchanged by the two parties. Two mechanisms are defined for referencing the  
884 `EncryptedKey`.

885

886 When referencing the `EncryptedKey` within the same message that contains the  
887 `<xenc:EncryptedKey>` element, the `<ds:KeyInfo>` element of the referencing construct MUST  
888 contain a `<wsse:SecurityTokenReference>`. The `<wsse:SecurityTokenReference>` element  
889 MUST contain a `<wsse:Reference>` element.

890

891 The URI attribute value of the `<wsse:Reference>` element MUST be set to the value of the ID attribute  
892 of the referenced `<xenc:EncryptedKey>` element that contains the `EncryptedKey`.

893 When referencing the `EncryptedKey` in a message that does not contain the `<xenc:EncryptedKey>`  
894 element, the `<ds:KeyInfo>` element of the referencing construct MUST contain a  
895 `<wsse:SecurityTokenReference>`. The `<wsse:SecurityTokenReference>` element MUST  
896 contain a `<wsse:KeyIdentifier>` element. The `EncodingType` attribute SHOULD be set to  
897 `#Base64Binary`. Other encoding types MAY be specified if agreed on by all parties. The  
898 `wsse11:TokenType` attribute MUST be set to

```
899 http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
900 1.1#EncryptedKey. The identifier for a <xenc:EncryptedKey> token is defined as the SHA1 of the
901 raw (pre-base64 encoding) octets specified in the <xenc:CipherValue> element of the referenced
902 <xenc:EncryptedKey> token. This value is encoded as indicated in the <wsse:KeyIdentifier>
903 reference. The <wsse:ValueType> attribute of <wsse:KeyIdentifier> MUST be set to
904 http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-
905 1.1#EncryptedKeySHA1.
```

---

## 906 8 Signatures

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

910  
911 Demonstrating knowledge of a confirmation key associated with a token key-claim confirms the  
912 accompanying token claims. Knowledge of a confirmation key may be demonstrated by using that key to  
913 create an XML Signature, for example. The relying party's acceptance of the claims may depend on its  
914 confidence in the token. Multiple tokens may contain a key-claim for a signature and may be referenced  
915 from the signature using a `<wsse:SecurityTokenReference>`. A key-claim may be an X.509  
916 Certificate token, or a Kerberos service ticket token to give two examples.

917  
918 Because of the mutability of some [SOAP](#) headers, producers SHOULD NOT use the *Enveloped*  
919 *Signature Transform* defined in [XML Signature](#). Instead, messages SHOULD explicitly include the  
920 elements to be signed. Similarly, producers SHOULD NOT use the *Enveloping Signature* defined in [XML](#)  
921 [Signature](#) [XMLSIG].

922  
923 This specification allows for multiple signatures and signature formats to be attached to a message, each  
924 referencing different, even overlapping, parts of the message. This is important for many distributed  
925 applications where messages flow through multiple processing stages. For example, a producer may  
926 submit an order that contains an orderID header. The producer signs the orderID header and the body of  
927 the request (the contents of the order). When this is received by the order processing sub-system, it may  
928 insert a shippingID into the header. The order sub-system would then sign, at a minimum, the orderID  
929 and the shippingID, and possibly the body as well. Then when this order is processed and shipped by the  
930 shipping department, a shippedInfo header might be appended. The shipping department would sign, at  
931 a minimum, the shippedInfo and the shippingID and possibly the body and forward the message to the  
932 billing department for processing. The billing department can verify the signatures and determine a valid  
933 chain of trust for the order, as well as who authorized each step in the process.

934  
935 All compliant implementations MUST be able to support the [XML Signature](#) standard.

### 936 8.1 Algorithms

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

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

940

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

941

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

943

Algorithm Type	Algorithm	Algorithm URI
Transform	SOAP Message Normalization	<a href="http://www.w3.org/TR/soap12-n11n/">http://www.w3.org/TR/soap12-n11n/</a>

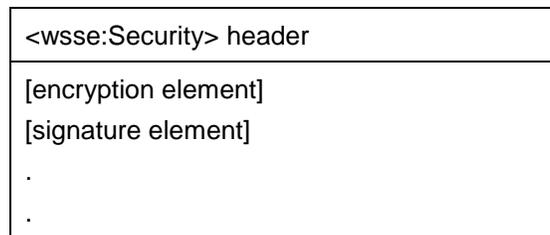
944

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

947

948 Finally, if a producer wishes to sign a message before encryption, then following the ordering rules laid  
 949 out in section 5, "Security Header", they SHOULD first prepend the signature element to the  
 950 <wsse:Security> header, and then prepend the encryption element, resulting in a <wsse:Security>  
 951 header that has the encryption element first, followed by the signature element:

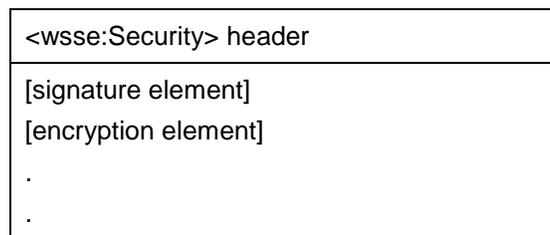
952



953

954 Likewise, if a producer wishes to sign a message after encryption, they SHOULD first prepend the  
 955 encryption element to the <wsse:Security> header, and then prepend the signature element. This  
 956 will result in a <wsse:Security> header that has the signature element first, followed by the encryption  
 957 element:

958



959

960 The XML Digital Signature WG has defined two canonicalization algorithms: XML Canonicalization and  
 961 Exclusive XML Canonicalization. To prevent confusion, the first is also called Inclusive Canonicalization.  
 962 Neither one solves all possible problems that can arise. The following informal discussion is intended to  
 963 provide guidance on the choice of which one to use

964 in particular circumstances. For a more detailed and technically precise discussion of these issues see:  
 965 [\[XML-C14N\]](#) and [\[EXCC14N\]](#).

966

967 There are two problems to be avoided. On the one hand, XML allows documents to be changed in  
 968 various ways and still be considered equivalent. For example, duplicate namespace declarations can be  
 969 removed or created. As a result, XML tools make these kinds of changes freely when processing XML.  
 970 Therefore, it is vital that these equivalent forms match the same signature.

971

972 On the other hand, if the signature simply covers something like `xx:foo`, its meaning may change if `xx` is  
973 redefined. In this case the signature does not prevent tampering. It might be thought that the problem  
974 could be solved by expanding all the values in line. Unfortunately, there are mechanisms like XPATH  
975 which consider `xx="http://example.com/"`; to be different from `yy="http://example.com/"`; even though both  
976 `xx` and `yy` are bound to the same namespace.

977 The fundamental difference between the Inclusive and Exclusive Canonicalization is the namespace  
978 declarations which are placed in the output. Inclusive Canonicalization copies all the declarations that are  
979 currently in force, even if they are defined outside of the scope of the signature. It also copies any `xml:`  
980 attributes that are in force, such as `xml:lang` or `xml:base`. This guarantees that all the declarations  
981 you might make use of will be unambiguously specified. The problem with this is that if the signed XML is  
982 moved into another XML document which has other declarations, the Inclusive Canonicalization will copy  
983 them and the signature will be invalid. This can even happen if you simply add an attribute in a different  
984 namespace to the surrounding context.

985

986 Exclusive Canonicalization tries to figure out what namespaces you are actually using and just copies  
987 those. Specifically, it copies the ones that are "visibly used", which means the ones that are a part of the  
988 XML syntax. However, it does not look into attribute values or element content, so the namespace  
989 declarations required to process these are not copied. For example

990 if you had an attribute like `xx:foo="yy:bar"` it would copy the declaration for `xx`, but not `yy`. (This can even  
991 happen without your knowledge because XML processing tools might add `xsi:type` if you use a  
992 schema subtype.) It also does not copy the `xml:` attributes that are declared outside the scope of the  
993 signature.

994

995 Exclusive Canonicalization allows you to create a list of the namespaces that must be declared, so that it  
996 will pick up the declarations for the ones that are not visibly used. The only problem is that the software  
997 doing the signing must know what they are. In a typical SOAP software environment, the security code  
998 will typically be unaware of all the namespaces being used by the application in the message body that it  
999 is signing.

1000

1001 Exclusive Canonicalization is useful when you have a signed XML document that you wish to insert into  
1002 other XML documents. A good example is a signed SAML assertion which might be inserted as a XML  
1003 Token in the security header of various SOAP messages. The Issuer who signs the assertion will be  
1004 aware of the namespaces being used and able to construct the list. The use of Exclusive Canonicalization  
1005 will insure the signature verifies correctly every time.

1006 Inclusive Canonicalization is useful in the typical case of signing part or all of the SOAP body in  
1007 accordance with this specification. This will insure all the declarations fall under the signature, even  
1008 though the code is unaware of what namespaces are being used. At the same time, it is less likely that  
1009 the signed data (and signature element) will be inserted in some other XML document. Even if this is  
1010 desired, it still may not be feasible for other reasons, for example there may be Id's with the same value  
1011 defined in both XML documents.

1012

1013 In other situations it will be necessary to study the requirements of the application and the detailed  
1014 operation of the canonicalization methods to determine which is appropriate.

1015 This section is non-normative.

## 1016 **8.2 Signing Messages**

1017 The `<wsse:Security>` header block MAY be used to carry a signature compliant with the [XML](#)  
1018 [Signature](#) specification within a [SOAP](#) Envelope for the purpose of signing one or more elements in the  
1019 [SOAP](#) Envelope. Multiple signature entries MAY be added into a single [SOAP](#) Envelope within one  
1020 `<wsse:Security>` header block. Producers SHOULD sign all important elements of the message, and

1021 careful thought must be given to creating a signing policy that requires signing of parts of the message  
1022 that might legitimately be altered in transit.

1023

1024 SOAP applications MUST satisfy the following conditions:

1025

1026 • A compliant implementation MUST be capable of processing the required elements defined in the  
1027 XML Signature specification.

1028 • To add a signature to a `<wsse:Security>` header block, a `<ds:Signature>` element  
1029 conforming to the XML Signature specification MUST be prepended to the existing content of the  
1030 `<wsse:Security>` header block, in order to indicate to the receiver the correct order of  
1031 operations. All the `<ds:Reference>` elements contained in the signature SHOULD refer to a  
1032 resource within the enclosing SOAP envelope as described in the XML Signature specification.  
1033 However, since the SOAP message exchange model allows intermediate applications to modify  
1034 the Envelope (add or delete a header block; for example), XPath filtering does not always result  
1035 in the same objects after message delivery. Care should be taken in using XPath filtering so that  
1036 there is no unintentional validation failure due to such modifications.

1037 • The problem of modification by intermediaries (especially active ones) is applicable to more than  
1038 just XPath processing. Digital signatures, because of canonicalization and digests, present  
1039 particularly fragile examples of such relationships. If overall message processing is to remain  
1040 robust, intermediaries must exercise care that the transformation algorithms used do not affect  
1041 the validity of a digitally signed component.

1042 • Due to security concerns with namespaces, this specification strongly RECOMMENDS the use of  
1043 the "Exclusive XML Canonicalization" algorithm or another canonicalization algorithm that  
1044 provides equivalent or greater protection.

1045 • For processing efficiency it is RECOMMENDED to have the signature added and then the  
1046 security token prepended so that a processor can read and cache the token before it is used.

## 1047 8.3 Signing Tokens

1048 It is often desirable to sign security tokens that are included in a message or even external to the  
1049 message. The XML Signature specification provides several common ways for referencing information to  
1050 be signed such as URIs, IDs, and XPath, but some token formats may not allow tokens to be referenced  
1051 using URIs or IDs and XPaths may be undesirable in some situations.

1052 This specification allows different tokens to have their own unique reference mechanisms which are  
1053 specified in their profile as extensions to the `<wsse:SecurityTokenReference>` element. This  
1054 element provides a uniform referencing mechanism that is guaranteed to work with all token formats.  
1055 Consequently, this specification defines a new reference option for XML Signature: the STR Dereference  
1056 Transform.

1057

1058 This transform is specified by the URI `#STR-Transform` and when applied to a  
1059 `<wsse:SecurityTokenReference>` element it means that the output is the token referenced by the  
1060 `<wsse:SecurityTokenReference>` element not the element itself.

1061

1062 As an overview the processing model is to echo the input to the transform except when a  
1063 `<wsse:SecurityTokenReference>` element is encountered. When one is found, the element is not  
1064 echoed, but instead, it is used to locate the token(s) matching the criteria and rules defined by the  
1065 `<wsse:SecurityTokenReference>` element and echo it (them) to the output. Consequently, the  
1066 output of the transformation is the resultant sequence representing the input with any  
1067 `<wsse:SecurityTokenReference>` elements replaced by the referenced security token(s) matched.

1068

1069 The following illustrates an example of this transformation which references a token contained within the  
1070 message envelope:

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

```
...
<wsse:SecurityTokenReference wsu:Id="Str1">
  ...
</wsse:SecurityTokenReference>
...
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    ...
    <ds:Reference URI="#Str1">
      <ds:Transforms>
        <ds:Transform
          Algorithm="...#STR-Transform">
          <wsse:TransformationParameters>
            <ds:CanonicalizationMethod
              Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315" />
          </wsse:TransformationParameters>
        </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod Algorithm=
        "http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue></ds:SignatureValue>
</ds:Signature>
...
```

1099

1100 The following describes the attributes and elements listed in the example above:

1101

1102 */wsse:TransformationParameters*

1103 This element is used to wrap parameters for a transformation allows elements even from the XML  
1104 Signature namespace.

1105

1106 */wsse:TransformationParameters/ds:Canonicalization*

1107 This specifies the canonicalization algorithm to apply to the selected data.

1108

1109 */wsse:TransformationParameters/{any}*

1110 This is an extensibility mechanism to allow different (extensible) parameters to be specified in the  
1111 future. Unrecognized parameters SHOULD cause a fault.

1112

1113 */wsse:TransformationParameters/@{any}*

1114 This is an extensibility mechanism to allow additional attributes, based on schemas, to be added  
1115 to the element in the future. Unrecognized attributes SHOULD cause a fault.

1116

1117 The following is a detailed specification of the transformation. The algorithm is identified by the URI:  
1118 #STR-Transform.

1119

1120 Transform Input:

- 1121 • The input is a node set. If the input is an octet stream, then it is automatically parsed; cf. XML  
1122 Digital Signature [XMLSIG].

1123 Transform Output:

- 1124 • The output is an octet stream.

1125 Syntax:

- 1126 • The transform takes a single mandatory parameter, a `<ds:CanonicalizationMethod>`  
1127 element, which is used to serialize the output node set. Note, however, that the output may not be  
1128 strictly in canonical form, per the canonicalization algorithm; however, the output is canonical, in  
1129 the sense that it is unambiguous. However, because of syntax requirements in the XML  
1130 Signature definition, this parameter MUST be wrapped in a  
1131 `<wsse:TransformationParameters>` element.  
1132 •

1133 Processing Rules:

- 1134 • Let N be the input node set.
- 1135 • Let R be the set of all `<wsse:SecurityTokenReference>` elements in N.
- 1136 • For each  $R_i$  in R, let  $D_i$  be the result of dereferencing  $R_i$ .
- 1137 • If  $D_i$  cannot be determined, then the transform MUST signal a failure.
- 1138 • If  $D_i$  is an XML security token (e.g., a SAML assertion or a `<wsse:BinarySecurityToken>`  
1139 element), then let  $R_i'$  be  $D_i$ . Otherwise,  $D_i$  is a raw binary security token; i.e., an octet stream. In  
1140 this case, let  $R_i'$  be a node set consisting of a `<wsse:BinarySecurityToken>` element,  
1141 utilizing the same namespace prefix as the `<wsse:SecurityTokenReference>` element  $R_i$ ,  
1142 with no `EncodingType` attribute, a `ValueType` attribute identifying the content of the security  
1143 token, and text content consisting of the binary-encoded security token, with no white space.
- 1144 • Finally, employ the canonicalization method specified as a parameter to the transform to serialize  
1145 N to produce the octet stream output of this transform; but, in place of any dereferenced  
1146 `<wsse:SecurityTokenReference>` element  $R_i$  and its descendants, process the  
1147 dereferenced node set  $R_i'$  instead. During this step, canonicalization of the replacement node set  
1148 MUST be augmented as follows:
  - 1149 ○ Note: A namespace declaration `xmlns=""` MUST be emitted with every apex element  
1150 that has no namespace node declaring a value for the default namespace; cf. XML  
1151 Decryption Transform.

1152 Note: Per the processing rules above, any `<wsse:SecurityTokenReference>` element is  
1153 effectively replaced by the referenced `<wsse:BinarySecurityToken>` element and then the  
1154 `<wsse:BinarySecurityToken>` is canonicalized in that context. Each  
1155 `<wsse:BinarySecurityToken>` needs to be complete in a given context, so any necessary  
1156 namespace declarations that are not present on an ancestor element will need to be added to the  
1157 `<wsse:BinarySecurityToken>` element prior to canonicalization.

1159 Signing a `<wsse:SecurityTokenReference>` (STR) element provides authentication and  
1160 integrity protection of only the STR and not the referenced security token (ST). If signing the ST is  
1161 the intended behavior, the STR Dereference Transform (STRDT) may be used which replaces  
1162 the STR with the ST for digest computation, effectively protecting the ST and not the STR. If  
1163 protecting both the ST and the STR is desired, you may sign the STR twice, once using the  
1164 STRDT and once not using the STRDT.

1166 The following table lists the full URI for each URI fragment referred to in the specification.

1167

URI Fragment	Full URI
#Base64Binary	<a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary</a>
#STR-Transform	<a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STRTransform">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STRTransform</a>

## 1168 8.4 Signature Validation

1169 The validation of a `<ds:Signature>` element inside an `<wsse:Security>` header block MUST fail if:

- 1170 • the syntax of the content of the element does not conform to this specification, or
- 1171 • the validation of the [signature](#) contained in the element fails according to the core validation of the
- 1172 [XML Signature](#) specification [XMLSIG], or
- 1173 • the application applying its own validation policy rejects the message for some reason (e.g., the
- 1174 [signature](#) is created by an untrusted key – verifying the previous two steps only performs
- 1175 cryptographic validation of the [signature](#)).

1176

1177 If the validation of the signature element fails, applications MAY report the failure to the producer using

1178 the fault codes defined in [Section 12 Error Handling](#).

1179

1180 The signature validation shall additionally adhere to the rules defines in signature confirmation section

1181 below, if the initiator desires signature confirmation:

## 1182 8.5 Signature Confirmation

1183 In the general model, the initiator uses XML Signature constructs to represent message parts of the

1184 request that were signed. The manifest of signed SOAP elements is contained in the `<ds:Signature>`

1185 element which in turn is placed inside the `<wsse:Security>` header. The `<ds:Signature>` element

1186 of the request contains a `<ds:SignatureValue>`. This element contains a base64 encoded value

1187 representing the actual digital signature. In certain situations it is desirable that initiator confirms that the

1188 message received was generated in response to a message it initiated in its unaltered form. This helps

1189 prevent certain forms of attack. This specification introduces a `<wsse11:SignatureConfirmation>`

1190 element to address this necessity.

1191

1192 Compliant responder implementations that support signature confirmation, MUST include a

1193 `<wsse11:SignatureConfirmation>` element inside the `<wsse:Security>` header of the

1194 associated response message for every `<ds:Signature>` element that is a direct child of the

1195 `<wsse:Security>` header block in the originating message. The responder MUST include the contents

1196 of the `<ds:SignatureValue>` element of the request signature as the value of the `@Value` attribute of

1197 the `<wsse11:SignatureConfirmation>` element. The `<wsse11:SignatureConfirmation>`

1198 element MUST be included in the message signature of the associated response message.

1199

1200 If the associated originating signature is received in encrypted form then the corresponding

1201 `<wsse11:SignatureConfirmation>` element SHOULD be encrypted to protect the original signature

1202 and keys.

1203

1204 The schema outline for this element is as follows:

1205

```
1206 <wsse11:SignatureConfirmation wsu:Id="..." Value="..." />
```

1207

1208 */wsse11:SignatureConfirmation*

1209 This element indicates that the responder has processed the signature in the request. When this

1210 element is not present in a response the initiator SHOULD interpret that the responder is not

1211 compliant with this functionality.

1212

1213 */wsse11:SignatureConfirmation/@wsu:Id*

1214 Identifier to be used when referencing this element in the `<ds:SignedInfo>` reference list of the  
1215 signature of the associated response message. This attribute **MUST** be present so that un-  
1216 ambiguous references can be made to this `<wsse11:SignatureConfirmation>` element.

1217  
1218 */wsse11:SignatureConfirmation/@Value*

1219 This optional attribute contains the contents of a `<ds:SignatureValue>` copied from the  
1220 associated request. If the request was not signed, then this attribute **MUST NOT** be present. If  
1221 this attribute is specified with an empty value, the initiator **SHOULD** interpret this as incorrect  
1222 behavior and process accordingly. When this attribute is not present, the initiator **SHOULD**  
1223 interpret this to mean that the response is based on a request that was not signed.

## 1224 8.5.1 Response Generation Rules

1225 Conformant responders **MUST** include at least one `<wsse11:SignatureConfirmation>` element in  
1226 the `<wsse:Security>` header in any response(s) associated with requests. That is, the normal  
1227 messaging patterns are not altered.

1228 For every response message generated, the responder **MUST** include a  
1229 `<wsse11:SignatureConfirmation>` element for every `<ds:Signature>` element it processed from  
1230 the original request message. The `Value` attribute **MUST** be set to the exact value of the  
1231 `<ds:SignatureValue>` element of the corresponding `<ds:Signature>` element. If no  
1232 `<ds:Signature>` elements are present in the original request message, the responder **MUST** include  
1233 exactly one `<wsse11:SignatureConfirmation>` element. The `Value` attribute of the  
1234 `<wsse11:SignatureConfirmation>` element **MUST NOT** be present. The responder **MUST** include  
1235 all `<wsse11:SignatureConfirmation>` elements in the message signature of the response  
1236 message(s). If the `<ds:Signature>` element corresponding to a  
1237 `<wsse11:SignatureConfirmation>` element was encrypted in the original request message, the  
1238 `<wsse11:SignatureConfirmation>` element **SHOULD** be encrypted for the recipient of the response  
1239 message(s).

## 1240 8.5.2 Response Processing Rules

1241 The signature validation shall additionally adhere to the following processing guidelines, if the initiator  
1242 desires signature confirmation:

- 1243 • If a response message does not contain a `<wsse11:SignatureConfirmation>` element  
1244 inside the `<wsse:Security>` header, the initiator **SHOULD** reject the response message.
- 1245 • If a response message does contain a `<wsse11:SignatureConfirmation>` element inside  
1246 the `<wsse:Security>` header but `@Value` attribute is not present on  
1247 `<wsse11:SignatureConfirmation>` element, and the associated request message did  
1248 include a `<ds:Signature>` element, the initiator **SHOULD** reject the response message.
- 1249 • If a response message does contain a `<wsse11:SignatureConfirmation>` element inside  
1250 the `<wsse:Security>` header and the `@Value` attribute is present on the  
1251 `<wsse11:SignatureConfirmation>` element, but the associated request did not include a  
1252 `<ds:Signature>` element, the initiator **SHOULD** reject the response message.
- 1253 • If a response message does contain a `<wsse11:SignatureConfirmation>` element inside  
1254 the `<wsse:Security>` header, and the associated request message did include a  
1255 `<ds:Signature>` element and the `@Value` attribute is present but does not match the stored  
1256 signature value of the associated request message, the initiator **SHOULD** reject the response  
1257 message.
- 1258 • If a response message does not contain a `<wsse11:SignatureConfirmation>` element  
1259 inside the `<wsse:Security>` header corresponding to each `<ds:Signature>` element or if  
1260 the `@Value` attribute present does not match the stored signature values of the associated  
1261 request message, the initiator **SHOULD** reject the response message.

## 1262 8.6 Example

1263 The following sample message illustrates the use of integrity and security tokens. For this example, only  
1264 the message body is signed.

1265

```
1266 <?xml version="1.0" encoding="utf-8"?>
1267 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:ds="...">
1268   <S11:Header>
1269     <wsse:Security>
1270       <wsse:BinarySecurityToken
1271         Value="http://docs.oasis-open.org/wss/2004/01/oasis-
1272 200401-wss-x509-token-profile-1.0#X509v3"
1273         EncodingType="...#Base64Binary"
1274         wsu:Id="X509Token">
1275         MIIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1276       </wsse:BinarySecurityToken>
1277     <ds:Signature>
1278       <ds:SignedInfo>
1279         <ds:CanonicalizationMethod Algorithm=
1280           "http://www.w3.org/2001/10/xml-exc-c14n#" />
1281         <ds:SignatureMethod Algorithm=
1282           "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
1283         <ds:Reference URI="#myBody">
1284           <ds:Transforms>
1285             <ds:Transform Algorithm=
1286               "http://www.w3.org/2001/10/xml-exc-c14n#" />
1287           </ds:Transforms>
1288           <ds:DigestMethod Algorithm=
1289             "http://www.w3.org/2000/09/xmldsig#sha1" />
1290           <ds:DigestValue>EULddytSo1...</ds:DigestValue>
1291         </ds:Reference>
1292       </ds:SignedInfo>
1293       <ds:SignatureValue>
1294       BL8jdfToEb11/vXcMZNNjPOV...
1295     </ds:SignatureValue>
1296     <ds:KeyInfo>
1297       <wsse:SecurityTokenReference>
1298         <wsse:Reference URI="#X509Token" />
1299       </wsse:SecurityTokenReference>
1300     </ds:KeyInfo>
1301   </ds:Signature>
1302 </wsse:Security>
1303 </S11:Header>
1304 <S11:Body wsu:Id="myBody">
1305   <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
1306     QQQ
1307   </tru:StockSymbol>
1308 </S11:Body>
1309 </S11:Envelope>
```

---

## 1310 9 Encryption

1311 This specification allows encryption of any combination of body blocks, header blocks, and any of these  
1312 sub-structures by either a common symmetric key shared by the producer and the recipient or a  
1313 symmetric key carried in the message in an encrypted form.

1314  
1315 In order to allow this flexibility, this specification leverages the [XML Encryption](#) standard. This  
1316 specification describes how the two elements `<xenc:ReferenceList>` and `<xenc:EncryptedKey>`  
1317 listed below and defined in [XML Encryption](#) can be used within the `<wsse:Security>` header block.  
1318 When a producer or an active intermediary encrypts portion(s) of a [SOAP](#) message using [XML Encryption](#)  
1319 it MUST prepend a sub-element to the `<wsse:Security>` header block. Furthermore, the encrypting  
1320 party MUST either prepend the sub-element to an existing `<wsse:Security>` header block for the  
1321 intended recipients or create a new `<wsse:Security>` header block and insert the sub-element. The  
1322 combined process of encrypting portion(s) of a message and adding one of these sub-elements is called  
1323 an encryption step hereafter. The sub-element MUST contain the information necessary for the recipient  
1324 to identify the portions of the message that it is able to decrypt.

1325  
1326 This specification additionally defines an element `<wsse11:EncryptedHeader>` for containing  
1327 encrypted SOAP header blocks. This specification RECOMMENDS an additional mechanism that uses  
1328 this element for encrypting SOAP header blocks that complies with SOAP processing guidelines while  
1329 preserving the confidentiality of attributes on the SOAP header blocks.

1330 All compliant implementations MUST be able to support the [XML Encryption](#) standard [XMLENC].

### 1331 9.1 xenc:ReferenceList

1332 The `<xenc:ReferenceList>` element from [XML Encryption](#) [XMLENC] MAY be used to create a  
1333 manifest of encrypted portion(s), which are expressed as `<xenc:EncryptedData>` elements within the  
1334 envelope. An element or element content to be encrypted by this encryption step MUST be replaced by a  
1335 corresponding `<xenc:EncryptedData>` according to [XML Encryption](#). All the  
1336 `<xenc:EncryptedData>` elements created by this encryption step SHOULD be listed in  
1337 `<xenc:DataReference>` elements inside one or more `<xenc:ReferenceList>` element.

1338  
1339 Although in [XML Encryption](#) [XMLENC], `<xenc:ReferenceList>` was originally designed to be used  
1340 within an `<xenc:EncryptedKey>` element (which implies that all the referenced  
1341 `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows that  
1342 `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>` MAY be  
1343 encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>` within individual  
1344 `<xenc:EncryptedData>`.

1345  
1346 A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the producer and  
1347 the recipient use a shared secret key. The following illustrates the use of this sub-element:

```
1348  
1349 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:ds="..."  
1350 xmlns:xenc="...">  
1351   <S11:Header>  
1352     <wsse:Security>  
1353       <xenc:ReferenceList>  
1354         <xenc:DataReference URI="#bodyID"/>  
1355     </wsse:Security>  
1356   </S11:Header>  
1357 </S11:Envelope>
```

```

1355         </xenc:ReferenceList>
1356     </wsse:Security>
1357 </S11:Header>
1358 <S11:Body>
1359     <xenc:EncryptedData Id="bodyID">
1360         <ds:KeyInfo>
1361             <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
1362         </ds:KeyInfo>
1363         <xenc:CipherData>
1364             <xenc:CipherValue>...</xenc:CipherValue>
1365         </xenc:CipherData>
1366     </xenc:EncryptedData>
1367 </S11:Body>
1368 </S11:Envelope>

```

## 1369 9.2 xenc:EncryptedKey

1370 When the encryption step involves encrypting elements or element contents within a [SOAP](#) envelope with  
1371 a symmetric key, which is in turn to be encrypted by the recipient's key and embedded in the message,  
1372 <xenc:EncryptedKey> MAY be used for carrying such an encrypted key. This sub-element MAY  
1373 contain a manifest, that is, an <xenc:ReferenceList> element, that lists the portions to be decrypted  
1374 with this key. The manifest MAY appear outside the <xenc:EncryptedKey> provided that the  
1375 corresponding xenc:EncryptedData

1376 elements contain <xenc:KeyInfo> elements that reference the <xenc:EncryptedKey> element.. An  
1377 element or element content to be encrypted by this encryption step MUST be replaced by a  
1378 corresponding <xenc:EncryptedData> according to [XML Encryption](#). All the  
1379 <xenc:EncryptedData> elements created by this encryption step SHOULD be listed in the  
1380 <xenc:ReferenceList> element inside this sub-element.

1381  
1382 This construct is useful when encryption is done by a randomly generated symmetric key that is in turn  
1383 encrypted by the recipient's public key. The following illustrates the use of this element:

```

1384
1385 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:ds="..."
1386 xmlns:xenc="...">
1387     <S11:Header>
1388         <wsse:Security>
1389             <xenc:EncryptedKey>
1390                 ...
1391                 <ds:KeyInfo>
1392                     <wsse:SecurityTokenReference>
1393                         <ds:X509IssuerSerial>
1394                             <ds:X509IssuerName>
1395                                 DC=ACMECorp, DC=com
1396                             </ds:X509IssuerName>
1397                         <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
1398                     </ds:X509IssuerSerial>
1399                 </wsse:SecurityTokenReference>
1400             </ds:KeyInfo>
1401             ...
1402             </xenc:EncryptedKey>
1403         ...
1404     </wsse:Security>
1405 </S11:Header>
1406 <S11:Body>
1407     <xenc:EncryptedData Id="bodyID">
1408         <xenc:CipherData>
1409             <xenc:CipherValue>...</xenc:CipherValue>
1410         </xenc:CipherData>

```

```
1411     </xenc:EncryptedData>
1412     </S11:Body>
1413 </S11:Envelope>
```

1414

1415 While XML Encryption specifies that `<xenc:EncryptedKey>` elements MAY be specified in  
1416 `<xenc:EncryptedData>` elements, this specification strongly RECOMMENDS that  
1417 `<xenc:EncryptedKey>` elements be placed in the `<wsse:Security>` header.

## 1418 9.3 Encrypted Header

1419 In order to be compliant with SOAP mustUnderstand processing guidelines and to prevent disclosure of  
1420 information contained in attributes on a SOAP header block, this specification introduces an  
1421 `<wsse11:EncryptedHeader>` element. This element contains exactly one `<xenc:EncryptedData>`  
1422 element. This specification RECOMMENDS the use of `<wsse11:EncryptedHeader>` element for  
1423 encrypting SOAP header blocks.

## 1424 9.4 Processing Rules

1425 Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the [XML](#)  
1426 [Encryption](#) specification. An encrypted [SOAP](#) envelope MUST still be a valid [SOAP](#) envelope. The  
1427 message creator MUST NOT encrypt the `<S11:Header>`, `<S12:Header>`, `<S11:Envelope>`,  
1428 `<S12:Envelope>`, or `<S11:Body>`, `<S12:Body>` elements but MAY encrypt child elements of either  
1429 the `<S11:Header>`, `<S12:Header>` and `<S11:Body>` or `<S12:Body>` elements. Multiple steps of  
1430 encryption MAY be added into a single `<wsse:Security>` header block if they are targeted for the  
1431 same recipient.

1432

1433 When an element or element content inside a [SOAP](#) envelope (e.g. the contents of the `<S11:Body>` or  
1434 `<S12:Body>` elements) are to be encrypted, it MUST be replaced by an `<xenc:EncryptedData>`,  
1435 according to [XML Encryption](#) and it SHOULD be referenced from the `<xenc:ReferenceList>` element  
1436 created by this encryption step. If the target of reference is an `EncryptedHeader` as defined in section  
1437 9.3 above, see processing rules defined in section 9.5.3 Encryption using `EncryptedHeader` and section  
1438 9.5.4 Decryption of `EncryptedHeader` below.

### 1439 9.4.1 Encryption

1440 The general steps (non-normative) for creating an encrypted [SOAP](#) message in compliance with this  
1441 specification are listed below (note that use of `<xenc:ReferenceList>` is RECOMMENDED.  
1442 Additionally, if the target of encryption is a SOAP header, processing rules defined in section 9.5.3  
1443 SHOULD be used).

- 1444 • Create a new [SOAP](#) envelope.
- 1445 • Create a `<wsse:Security>` header
- 1446 • When an `<xenc:EncryptedKey>` is used, create a `<xenc:EncryptedKey>` sub-element of  
1447 the `<wsse:Security>` element. This `<xenc:EncryptedKey>` sub-element SHOULD contain  
1448 an `<xenc:ReferenceList>` sub-element, containing a `<xenc:DataReference>` to each  
1449 `<xenc:EncryptedData>` element that was encrypted using that key.
- 1450 • Locate data items to be encrypted, i.e., XML elements, element contents within the target [SOAP](#)  
1451 envelope.
- 1452 • Encrypt the data items as follows: For each XML element or element content within the target  
1453 [SOAP](#) envelope, encrypt it according to the processing rules of the [XML Encryption](#) specification  
1454 [XMLENC]. Each selected original element or element content MUST be removed and replaced  
1455 by the resulting `<xenc:EncryptedData>` element.

- 1456
- 1457
- 1458
- 1459
- 1460
- 1461
- 1462
- 1463
- The optional `<ds:KeyInfo>` element in the `<xenc:EncryptedData>` element MAY reference another `<ds:KeyInfo>` element. Note that if the encryption is based on an attached security token, then a `<wsse:SecurityTokenReference>` element SHOULD be added to the `<ds:KeyInfo>` element to facilitate locating it.
  - Create an `<xenc:DataReference>` element referencing the generated `<xenc:EncryptedData>` elements. Add the created `<xenc:DataReference>` element to the `<xenc:ReferenceList>`.
  - Copy all non-encrypted data.

## 1464 9.4.2 Decryption

1465 On receiving a SOAP envelope containing encryption header elements, for each encryption header  
1466 element the following general steps should be processed (this section is non-normative. Additionally, if  
1467 the target of reference is an `EncryptedHeader`, processing rules as defined in section 9.5.4 below  
1468 SHOULD be used):

- 1469
1. Identify any decryption keys that are in the recipient's possession, then identifying any message  
1470 elements that it is able to decrypt.
  - 1471 2. Locate the `<xenc:EncryptedData>` items to be decrypted (possibly using the  
1472 `<xenc:ReferenceList>`).
  - 1473 3. Decrypt them as follows:  
1474
    - 1475 a. For each element in the target SOAP envelope, decrypt it according to the processing  
1476 rules of the XML Encryption specification and the processing rules listed above.
    - 1477 b. If the decryption fails for some reason, applications MAY report the failure to the producer  
1478 using the fault code defined in Section 12 Error Handling of this specification.
    - 1479 c. It is possible for overlapping portions of the SOAP message to be encrypted in such a  
1480 way that they are intended to be decrypted by SOAP nodes acting in different Roles. In  
1481 this case, the `<xenc:ReferenceList>` or `<xenc:EncryptedKey>` elements  
1482 identifying these encryption operations will necessarily appear in different  
1483 `<wsse:Security>` headers. Since SOAP does not provide any means of specifying the  
1484 order in which different Roles will process their respective headers, this order is not  
1485 specified by this specification and can only be determined by a prior agreement.

## 1486 9.4.3 Encryption with EncryptedHeader

1487 When it is required that an entire SOAP header block including the top-level element and its attributes be  
1488 encrypted, the original header block SHOULD be replaced with a `<wsse11:EncryptedHeader>`  
1489 element. The `<wsse11:EncryptedHeader>` element MUST contain the `<xenc:EncryptedData>`  
1490 produced by encrypting the header block. A `wsu:Id` attribute MAY be added to the  
1491 `<wsse11:EncryptedHeader>` element for referencing. If the referencing `<wsse:Security>` header  
1492 block defines a value for the `<S12:mustUnderstand>` or `<S11:mustUnderstand>` attribute, that  
1493 attribute and associated value MUST be copied to the `<wsse11:EncryptedHeader>` element. If the  
1494 referencing `<wsse:Security>` header block defines a value for the `S12:role` or `S11:actor` attribute,  
1495 that attribute and associated value MUST be copied to the `<wsse11:EncryptedHeader>` element. If  
1496 the referencing `<wsse:Security>` header block defines a value for the `S12:relay` attribute, that  
1497 attribute and associated value MUST be copied to the `<wsse11:EncryptedHeader>` element.

1498

1499 Any header block can be replaced with a corresponding `<wsse11:EncryptedHeader>` header block.  
1500 This includes `<wsse:Security>` header blocks. (In this case, obviously if the encryption operation is  
1501 specified in the same security header or in a security header targeted at a node which is reached after the  
1502 node targeted by the `<wsse11:EncryptedHeader>` element, the decryption will not occur.)

1503

1504 In addition, <wssell:EncryptedHeader> header blocks can be super-encrypted and replaced by  
1505 other <wssell:EncryptedHeader> header blocks (for wrapping/tunneling scenarios). Any  
1506 <wsse:Security> header that encrypts a header block targeted to a particular actor SHOULD be  
1507 targeted to that same actor, unless it is a security header.

## 1508 9.4.4 Processing an EncryptedHeader

1509 The processing model for <wssell:EncryptedHeader> header blocks is as follows:

- 1510 1. Resolve references to encrypted data specified in the <wsse:Security> header block targeted  
1511 at this node. For each reference, perform the following steps.
- 1512 2. If the referenced element does not have a qualified name of <wssell:EncryptedHeader>  
1513 then process as per section 9.4.2 Decryption and stop the processing steps here.
- 1514 3. Otherwise, extract the <xenc:EncryptedData> element from the  
1515 <wssell:EncryptedHeader> element.
- 1516 4. Decrypt the contents of the <xenc:EncryptedData> element as per section 9.4.2 Decryption  
1517 and replace the <wssell:EncryptedHeader> element with the decrypted contents.
- 1518 5. Process the decrypted header block as per SOAP processing guidelines.

1519

1520 Alternatively, a processor may perform a pre-pass over the encryption references in the  
1521 <wsse:Security> header:

- 1522 1. Resolve references to encrypted data specified in the <wsse:Security> header block targeted  
1523 at this node. For each reference, perform the following steps.
- 1524 2. If a referenced element has a qualified name of <wssell:EncryptedHeader> then replace the  
1525 <wssell:EncryptedHeader> element with the contained <xenc:EncryptedData> element  
1526 and if present copy the value of the wsu:Id attribute from the <wssell:EncryptedHeader>  
1527 element to the <xenc:EncryptedData> element.
- 1528 3. Process the <wsse:Security> header block as normal.

1529

1530 It should be noted that the results of decrypting a <wssell:EncryptedHeader> header block could be  
1531 another <wssell:EncryptedHeader> header block. In addition, the result MAY be targeted at a  
1532 different role than the role processing the <wssell:EncryptedHeader> header block.

## 1533 9.4.5 Processing the mustUnderstand attribute on EncryptedHeader

1534 If the S11:mustUnderstand or S12:mustUnderstand attribute is specified on the  
1535 <wssell:EncryptedHeader> header block, and is true, then the following steps define what it means  
1536 to "understand" the <wssell:EncryptedHeader> header block:

- 1537 1. The processor MUST be aware of this element and know how to decrypt and convert into the  
1538 original header block. This DOES NOT REQUIRE that the process know that it has the correct  
1539 keys or support the indicated algorithms.
- 1540 2. The processor MUST, after decrypting the encrypted header block, process the decrypted header  
1541 block according to the SOAP processing guidelines. The receiver MUST raise a fault if any  
1542 content required to adequately process the header block remains encrypted or if the decrypted  
1543 SOAP header is not understood and the value of the S12:mustUnderstand or  
1544 S11:mustUnderstand attribute on the decrypted header block is true. Note that in order to  
1545 comply with SOAP processing rules in this case, the processor must roll back any persistent  
1546 effects of processing the security header, such as storing a received token.

---

## 10 Security Timestamps

1547

1548 It is often important for the recipient to be able to determine the *freshness* of security semantics. In some  
1549 cases, security semantics may be so *stale* that the recipient may decide to ignore it.

1550 This specification does not provide a mechanism for synchronizing time. The assumption is that time is  
1551 trusted or additional mechanisms, not described here, are employed to prevent replay.

1552 This specification defines and illustrates time references in terms of the `xsd:dateTime` type defined in  
1553 XML Schema. It is RECOMMENDED that all time references use this type. All references MUST be in  
1554 UTC time. Implementations MUST NOT generate time instants that specify leap seconds. If, however,  
1555 other time types are used, then the `ValueType` attribute (described below) MUST be specified to indicate  
1556 the data type of the time format. Requestors and receivers SHOULD NOT rely on other applications  
1557 supporting time resolution finer than milliseconds.

1558

1559 The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and expiration times  
1560 of the security semantics in a message.

1561

1562 All times MUST be in UTC format as specified by the [XML Schema](#) type (`dateTime`). It should be noted  
1563 that times support time precision as defined in the [XML Schema](#) specification.

1564 The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and may only  
1565 be present at most once per header (that is, per [SOAP](#) actor/role).

1566

1567 The ordering within the element is as illustrated below. The ordering of elements in the  
1568 `<wsu:Timestamp>` element is fixed and MUST be preserved by intermediaries.

1569 The schema outline for the `<wsu:Timestamp>` element is as follows:

1570

```
1571 <wsu:Timestamp wsu:Id="...">  
1572   <wsu:Created ValueType="...">...</wsu:Created>  
1573   <wsu:Expires ValueType="...">...</wsu:Expires>  
1574   ...  
1575 </wsu:Timestamp>
```

1576

1577 The following describes the attributes and elements listed in the schema above:

1578

1579 */wsu:Timestamp*

This is the element for indicating security semantics timestamps.

1581

1582 */wsu:Timestamp/wsui:Created*

1583 This represents the [creation time](#) of the security semantics. This element is optional, but can only  
1584 be specified once in a `<wsu:Timestamp>` element. Within the SOAP processing model,  
1585 creation is the instant that the infoset is serialized for transmission. The creation time of the  
1586 message SHOULD NOT differ substantially from its transmission time. The difference in time  
1587 should be minimized.

1588

1589 */wsu:Timestamp/wsui:Expires*

1590 This element represents the [expiration](#) of the security semantics. This is optional, but can appear  
1591 at most once in a `<wsu:Timestamp>` element. Upon expiration, the requestor asserts that its  
1592 security semantics are no longer valid. It is strongly RECOMMENDED that recipients (anyone

1593 who processes this message) discard (ignore) any message whose security semantics have  
1594 passed their expiration. A Fault code (`wsu:MessageExpired`) is provided if the recipient wants  
1595 to inform the requestor that its security semantics were expired. A service MAY issue a Fault  
1596 indicating the security semantics have expired.

1597  
1598 */wsu:Timestamp/{any}*  
1599 This is an extensibility mechanism to allow additional elements to be added to the element.  
1600 Unrecognized elements SHOULD cause a fault.

1601  
1602 */wsu:Timestamp/@wsu:Id*  
1603 This optional attribute specifies an XML Schema ID that can be used to reference this element  
1604 (the timestamp). This is used, for example, to reference the timestamp in a XML Signature.

1605  
1606 */wsu:Timestamp/@{any}*  
1607 This is an extensibility mechanism to allow additional attributes to be added to the element.  
1608 Unrecognized attributes SHOULD cause a fault.

1609  
1610 The expiration is relative to the requestor's clock. In order to evaluate the expiration time, recipients need  
1611 to recognize that the requestor's clock may not be synchronized to the recipient's clock. The recipient,  
1612 therefore, MUST make an assessment of the level of trust to be placed in the requestor's clock, since the  
1613 recipient is called upon to evaluate whether the expiration time is in the past relative to the requestor's,  
1614 not the recipient's, clock. The recipient may make a judgment of the requestor's likely current clock time  
1615 by means not described in this specification, for example an out-of-band clock synchronization protocol.  
1616 The recipient may also use the [creation time](#) and the delays introduced by intermediate [SOAP](#) roles to  
1617 estimate the degree of clock skew.

1618  
1619 The following example illustrates the use of the `<wsu:Timestamp>` element and its content.

1620

```
1621 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">  
1622   <S11:Header>  
1623     <wsse:Security>  
1624       <wsu:Timestamp wsu:Id="timestamp">  
1625         <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>  
1626         <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>  
1627       </wsu:Timestamp>  
1628       ...  
1629     </wsse:Security>  
1630     ...  
1631   </S11:Header>  
1632   <S11:Body>  
1633     ...  
1634   </S11:Body>  
1635 </S11:Envelope>
```

1636

## 11 Extended Example

1637 The following sample message illustrates the use of security tokens, signatures, and encryption. For this  
1638 example, the timestamp and the message body are signed prior to encryption. The decryption  
1639 transformation is not needed as the signing/encryption order is specified within the <wsse:Security>  
1640 header.

1641

```
1642 (001) <?xml version="1.0" encoding="utf-8"?>
1643 (002) <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
1644 xmlns:xenc="..." xmlns:ds="...">
1645 (003)   <S11:Header>
1646 (004)     <wsse:Security>
1647 (005)       <wsu:Timestamp wsu:Id="T0">
1648 (006)         <wsu:Created>
1649 (007)           2001-09-13T08:42:00Z</wsu:Created>
1650 (008)       </wsu:Timestamp>
1651 (009)
1652 (010)       <wsse:BinarySecurityToken
1653             ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
1654 200401-wss-x509-token-profile-1.0#X509v3"
1655             wsu:Id="X509Token"
1656             EncodingType="...#Base64Binary">
1657 (011)         MIIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
1658 (012)       </wsse:BinarySecurityToken>
1659 (013)       <xenc:EncryptedKey>
1660 (014)         <xenc:EncryptionMethod Algorithm=
1661             "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
1662 (015)         <ds:KeyInfo>
1663             <wsse:SecurityTokenReference>
1664 (016)               <wsse:KeyIdentifier
1665                     EncodingType="...#Base64Binary"
1666                     ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
1667 200401-wss-x509-token-profile-1.0#X509v3">MIGfMa0GCSq...
1668 (017)               </wsse:KeyIdentifier>
1669             </wsse:SecurityTokenReference>
1670 (018)         </ds:KeyInfo>
1671 (019)         <xenc:CipherData>
1672 (020)           <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1673 (021)         </xenc:CipherValue>
1674 (022)       </xenc:CipherData>
1675 (023)       <xenc:ReferenceList>
1676 (024)         <xenc:DataReference URI="#enc1"/>
1677 (025)       </xenc:ReferenceList>
1678 (026)     </xenc:EncryptedKey>
1679 (027)     <ds:Signature>
1680 (028)       <ds:SignedInfo>
1681 (029)         <ds:CanonicalizationMethod
1682             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1683 (030)         <ds:SignatureMethod
1684             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
1685 (031)         <ds:Reference URI="#T0">
1686 (032)           <ds:Transforms>
1687 (033)             <ds:Transform
1688                 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
1689 (034)           </ds:Transforms>
1690 (035)         <ds:DigestMethod
1691             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```

1692 (036) <ds:DigestValue>LyLsF094hPi4wPU...
1693 (037) </ds:DigestValue>
1694 (038) </ds:Reference>
1695 (039) <ds:Reference URI="#body">
1696 (040) <ds:Transforms>
1697 (041) <ds:Transform
1698 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1699 (042) </ds:Transforms>
1700 (043) <ds:DigestMethod
1701 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
1702 (044) <ds:DigestValue>LyLsF094hPi4wPU...
1703 (045) </ds:DigestValue>
1704 (046) </ds:Reference>
1705 (047) </ds:SignedInfo>
1706 (048) <ds:SignatureValue>
1707 (049) Hp1ZkmFZ/2kQLXDJbchm5gK...
1708 (050) </ds:SignatureValue>
1709 (051) <ds:KeyInfo>
1710 (052) <wsse:SecurityTokenReference>
1711 (053) <wsse:Reference URI="#X509Token" />
1712 (054) </wsse:SecurityTokenReference>
1713 (055) </ds:KeyInfo>
1714 (056) </ds:Signature>
1715 (057) </wsse:Security>
1716 (058) </S11:Header>
1717 (059) <S11:Body wsu:Id="body">
1718 (060) <xenc:EncryptedData
1719 Type="http://www.w3.org/2001/04/xmlenc#Element"
1720 wsu:Id="enc1">
1721 (061) <xenc:EncryptionMethod
1722 Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
1723 (062) <xenc:CipherData>
1724 (063) <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
1725 (064) </xenc:CipherValue>
1726 (065) </xenc:CipherData>
1727 (066) </xenc:EncryptedData>
1728 (067) </S11:Body>
1729 (068) </S11:Envelope>

```

1730

1731 Let's review some of the key sections of this example:

1732 Lines (003)-(058) contain the SOAP message headers.

1733

1734 Lines (004)-(057) represent the `<wsse:Security>` header block. This contains the security-related  
1735 information for the message.

1736

1737 Lines (005)-(008) specify the timestamp information. In this case it indicates the creation time of the  
1738 security semantics.

1739

1740 Lines (010)-(012) specify a [security token](#) that is associated with the message. In this case, it specifies  
1741 an [X.509](#) certificate that is encoded as Base64. Line (011) specifies the actual Base64 encoding of the  
1742 certificate.

1743

1744 Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a  
1745 symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to encrypt the  
1746 key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the symmetric key. Lines  
1747 (019)-(022) specify the actual encrypted form of the symmetric key. Lines (023)-(025) identify the

1748 encryption block in the message that uses this symmetric key. In this case it is only used to encrypt the  
1749 body (Id="enc1").

1750

1751 Lines (027)-(056) specify the digital signature. In this example, the signature is based on the [X.509](#)  
1752 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039) references the  
1753 message body.

1754

1755 Lines (048)-(050) indicate the actual signature value – specified in Line (043).

1756

1757 Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the [X.509](#) certificate  
1758 included in the message. Line (053) provides a URI link to the Lines (010)-(012).

1759 The body of the message is represented by Lines (059)-(067).

1760

1761 Lines (060)-(066) represent the encrypted metadata and form of the body using [XML Encryption](#). Line  
1762 (060) indicates that the "element value" is being replaced and identifies this encryption. Line (061)  
1763 specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the actual cipher  
1764 text (i.e., the result of the encryption). Note that we don't include a reference to the key as the key  
1765 references this encryption – Line (024).

---

## 12 Error Handling

1766

1767 There are many circumstances where an *error* can occur while processing security information. For  
1768 example:

- 1769 • Invalid or unsupported type of security token, signing, or encryption
- 1770 • Invalid or unauthenticated or unauthenticatable security token
- 1771 • Invalid signature
- 1772 • Decryption failure
- 1773 • Referenced security token is unavailable
- 1774 • Unsupported namespace

1775

1776 If a service does not perform its normal operation because of the contents of the Security header, then  
1777 that MAY be reported using SOAP's Fault Mechanism. This specification does not mandate that faults be  
1778 returned as this could be used as part of a denial of service or cryptographic attack. We combine  
1779 signature and encryption failures to mitigate certain types of attacks.

1780

1781 If a failure is returned to a producer then the failure MUST be reported using the SOAP Fault  
1782 mechanism. The following tables outline the predefined security fault codes. The "unsupported" classes  
1783 of errors are as follows. Note that the reason text provided below is RECOMMENDED, but alternative  
1784 text MAY be provided if more descriptive or preferred by the implementation. The tables below are  
1785 defined in terms of SOAP 1.1. For SOAP 1.2, the Fault/Code/Value is `env:Sender` (as defined in SOAP  
1786 1.2) and the Fault/Code/Subcode/Value is the *faultcode* below and the Fault/Reason/Text is the  
1787 *faultstring* below.

1788

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

1789

1790 The "failure" class of errors are:

1791

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	wsse:FailedAuthentication

authorized	
The signature or decryption was invalid	wsse:FailedCheck
Referenced security token could not be retrieved	wsse:SecurityTokenUnavailable
The message has expired	wsse:MessageExpired

1792

---

## 1793 13 Security Considerations

1794 As stated in the Goals and Requirements section of this document, this specification is meant to provide  
1795 extensible framework and flexible syntax, with which one could implement various security mechanisms.  
1796 This framework and syntax by itself *does not provide any guarantee of security*. When implementing and  
1797 using this framework and syntax, one must make every effort to ensure that the result is not vulnerable to  
1798 any one of a wide range of attacks.

### 1799 13.1 General Considerations

1800 It is not feasible to provide a comprehensive list of security considerations for such an extensible set of  
1801 mechanisms. A complete security analysis **MUST** be conducted on specific solutions based on this  
1802 specification. Below we illustrate some of the security concerns that often come up with protocols of this  
1803 type, but we stress that this *is not an exhaustive list of concerns*.

- 1804 • freshness guarantee (e.g., the danger of replay, delayed messages and the danger of relying on  
1805 timestamps assuming secure clock synchronization)
- 1806 • proper use of digital signature and encryption (signing/encrypting critical parts of the message,  
1807 interactions between signatures and encryption), i.e., signatures on (content of) encrypted  
1808 messages leak information when in plain-text)
- 1809 • protection of security tokens (integrity)
- 1810 • certificate verification (including revocation issues)
- 1811 • the danger of using passwords without utmost protection (i.e. dictionary attacks against  
1812 passwords, replay, insecurity of password derived keys, ...)
- 1813 • the use of randomness (or strong pseudo-randomness)
- 1814 • interaction between the security mechanisms implementing this standard and other system  
1815 component
- 1816 • man-in-the-middle attacks
- 1817 • PKI attacks (i.e. identity mix-ups)

1818  
1819 There are other security concerns that one may need to consider in security protocols. The list above  
1820 should not be used as a "check list" instead of a comprehensive security analysis. The next section will  
1821 give a few details on some of the considerations in this list.

### 1822 13.2 Additional Considerations

#### 1823 13.2.1 Replay

1824 Digital signatures alone do not provide message authentication. One can record a signed message and  
1825 resend it (a replay attack). It is strongly **RECOMMENDED** that messages include digitally signed elements  
1826 to allow message recipients to detect replays of the message when the messages are exchanged via an  
1827 open network. These can be part of the message or of the headers defined from other **SOAP**  
1828 extensions. Four typical approaches are: Timestamp, Sequence Number, Expirations and Message  
1829 Correlation. Signed timestamps **MAY** be used to keep track of messages (possibly by caching the most  
1830 recent timestamp from a specific service) and detect replays of previous messages. It is  
1831 **RECOMMENDED** that timestamps be cached for a given period of time, as a guideline, a value of five  
1832 minutes can be used as a minimum to detect replays, and that timestamps older than that given period of  
1833 time set be rejected in interactive scenarios.

## 1834 13.2.2 Combining Security Mechanisms

1835 This specification defines the use of [XML Signature](#) and [XML Encryption](#) in [SOAP](#) headers. As one of the  
1836 building blocks for securing [SOAP](#) messages, it is intended to be used in conjunction with other security  
1837 techniques. Digital signatures need to be understood in the context of other security mechanisms and  
1838 possible threats to an entity.

1839  
1840 Implementers should also be aware of all the security implications resulting from the use of digital  
1841 signatures in general and [XML Signature](#) in particular. When building trust into an application based on a  
1842 digital signature there are other technologies, such as certificate evaluation, that must be incorporated,  
1843 but these are outside the scope of this document.

1844  
1845 As described in [XML Encryption](#), the combination of signing and encryption over a common data item  
1846 may introduce some cryptographic vulnerability. For example, encrypting digitally signed data, while  
1847 leaving the digital signature in the clear, may allow plain text guessing attacks.

## 1848 13.2.3 Challenges

1849 When digital signatures are used for verifying the claims pertaining to the sending entity, the producer  
1850 must demonstrate knowledge of the confirmation key. One way to achieve this is to use a challenge-  
1851 response type of protocol. Such a protocol is outside the scope of this document.

1852 To this end, the developers can attach timestamps, expirations, and sequences to messages.

## 1853 13.2.4 Protecting Security Tokens and Keys

1854 Implementers should be aware of the possibility of a token substitution attack. In any situation where a  
1855 digital signature is verified by reference to a token provided in the message, which specifies the key, it  
1856 may be possible for an unscrupulous producer to later claim that a different token, containing the same  
1857 key, but different information was intended.

1858 An example of this would be a user who had multiple X.509 certificates issued relating to the same key  
1859 pair but with different attributes, constraints or reliance limits. Note that the signature of the token by its  
1860 issuing authority does not prevent this attack. Nor can an authority effectively prevent a different authority  
1861 from issuing a token over the same key if the user can prove possession of the secret.

1862  
1863 The most straightforward counter to this attack is to insist that the token (or its unique identifying data) be  
1864 included under the signature of the producer. If the nature of the application is such that the contents of  
1865 the token are irrelevant, assuming it has been issued by a trusted authority, this attack may be ignored.  
1866 However because application semantics may change over time, best practice is to prevent this attack.

1867  
1868 Requestors should use digital signatures to sign security tokens that do not include signatures (or other  
1869 protection mechanisms) to ensure that they have not been altered in transit. It is strongly  
1870 RECOMMENDED that all relevant and immutable message content be signed by the producer. Receivers  
1871 SHOULD only consider those portions of the document that are covered by the producer's signature as  
1872 being subject to the security tokens in the message. Security tokens appearing in `<wsse:Security>`  
1873 header elements SHOULD be signed by their issuing authority so that message receivers can have  
1874 confidence that the security tokens have not been forged or altered since their issuance. It is strongly  
1875 RECOMMENDED that a message producer sign any `<wsse:SecurityToken>` elements that it is  
1876 confirming and that are not signed by their issuing authority.

1877 When a requester provides, within the request, a Public Key to be used to encrypt the response, it is  
1878 possible that an attacker in the middle may substitute a different Public Key, thus allowing the attacker to  
1879 read the response. The best way to prevent this attack is to bind the encryption key in some way to the  
1880 request. One simple way of doing this is to use the same key pair to sign the request as to encrypt the

1881 response. However, if policy requires the use of distinct key pairs for signing and encryption, then the  
1882 Public Key provided in the request should be included under the signature of the request.

### 1883 **13.2.5 Protecting Timestamps and Ids**

1884 In order to *trust* `wsu:Id` attributes and `<wsu:Timestamp>` elements, they SHOULD be signed using the  
1885 mechanisms outlined in this specification. This allows readers of the IDs and timestamps information to  
1886 be certain that the IDs and timestamps haven't been forged or altered in any way. It is strongly  
1887 RECOMMENDED that IDs and timestamp elements be signed.

### 1888 **13.2.6 Protecting against removal and modification of XML Elements**

1889 XML Signatures using Shorthand XPath References (AKA IDREF) protect against the removal and  
1890 modification of XML elements; but do not protect the location of the element within the XML Document.

1891  
1892 Whether or not this is a security vulnerability depends on whether the location of the signed data within its  
1893 surrounding context has any semantic import. This consideration applies to data carried in the SOAP  
1894 Body or the Header.

1895  
1896 Of particular concern is the ability to relocate signed data into a SOAP Header block which is unknown to  
1897 the receiver and marked `mustUnderstand="false"`. This could have the effect of causing the receiver to  
1898 ignore signed data which the sender expected would either be processed or result in the generation of a  
1899 MustUnderstand fault.

1900  
1901 A similar exploit would involve relocating signed data into a SOAP Header block targeted to a `S11:actor`  
1902 or `S12:role` other than that which the sender intended, and which the receiver will not process.

1903  
1904 While these attacks could apply to any portion of the message, their effects are most pernicious with  
1905 SOAP header elements which may not always be present, but must be processed whenever they appear.

1906  
1907 In the general case of XML Documents and Signatures, this issue may be resolved by signing the entire  
1908 XML Document and/or strict XML Schema specification and enforcement. However, because elements of  
1909 the SOAP message, particularly header elements, may be legitimately modified by SOAP intermediaries,  
1910 this approach is usually not appropriate. It is RECOMMENDED that applications signing any part of the  
1911 SOAP body sign the entire body.

1912  
1913 Alternatives countermeasures include (but are not limited to):

- 1914 • References using XPath transforms with Absolute Path expressions with checks performed by  
1915 the receiver that the URI and Absolute Path XPath expression evaluate to the digested nodeset.
- 1916 • A Reference using an XPath transform to include any significant location-dependent elements  
1917 and exclude any elements that might legitimately be removed, added, or altered by  
1918 intermediaries,
- 1919 • Using only References to elements with location-independent semantics,
- 1920 • Strict policy specification and enforcement regarding which message parts are to be signed. For  
1921 example:
  - 1922 ○ Requiring that the entire SOAP Body and all children of SOAP Header be signed,
  - 1923 ○ Requiring that SOAP header elements which are marked `MustUnderstand="false"`  
1924 and have signed descendants MUST include the `MustUnderstand` attribute under the  
1925 signature.

1926

1927 **13.2.7 Detecting Duplicate Identifiers**

1928 The <wsse:Security> processing SHOULD check for duplicate values from among the set of ID  
1929 attributes that it is aware of. The wsse:Security processing MUST generate a fault if a duplicate ID value  
1930 is detected.

1931

1932 This section is non-normative.

1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957

---

## 14 Interoperability Notes

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

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

This section is non-normative.

---

## 15 Privacy Considerations

- 1958
- 1959 In the context of this specification, we are only concerned with potential privacy violation by the security  
1960 elements defined here. Privacy of the content of the payload message is out of scope.
- 1961 Producers or sending applications should be aware that claims, as collected in security tokens, are  
1962 typically personal information, and should thus only be sent according to the producer's privacy policies.  
1963 Future standards may allow privacy obligations or restrictions to be added to this data. Unless such  
1964 standards are used, the producer must ensure by out-of-band means that the recipient is bound to  
1965 adhering to all restrictions associated with the data, and the recipient must similarly ensure by out-of-band  
1966 means that it has the necessary consent for its intended processing of the data.
- 1967
- 1968 If claim data are visible to intermediaries, then the policies must also allow the release to these  
1969 intermediaries. As most personal information cannot be released to arbitrary parties, this will typically  
1970 require that the actors are referenced in an identifiable way; such identifiable references are also typically  
1971 needed to obtain appropriate encryption keys for the intermediaries.
- 1972 If intermediaries add claims, they should be guided by their privacy policies just like the original  
1973 producers.
- 1974
- 1975 Intermediaries may also gain traffic information from a SOAP message exchange, e.g., who  
1976 communicates with whom at what time. Producers that use intermediaries should verify that releasing this  
1977 traffic information to the chosen intermediaries conforms to their privacy policies.
- 1978
- 1979 This section is non-normative.

---

## 16References

- 1980
- 1981 **[GLOSS]** Informational RFC 2828, "[Internet Security Glossary](#)," May 2000.
- 1982 **[KERBEROS]** J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5),"
- 1983 [RFC 1510](#), September 1993, <http://www.ietf.org/rfc/rfc1510.txt> .
- 1984 **[KEYWORDS]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," [RFC 2119](#), Harvard University, March 1997.
- 1985
- 1986 **[SHA-1]** FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce /
- 1987 National Institute of Standards and Technology.
- 1988 <http://www.itl.nist.gov/fipspubs/fip180-1.htm>
- 1989 **[SOAP11]** W3C Note, "[SOAP: Simple Object Access Protocol 1.1](#)," 08 May 2000.
- 1990 **[SOAP12]** W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework", 23
- 1991 June 2003.
- 1992 **[SOAPSEC]** W3C Note, "[SOAP Security Extensions: Digital Signature](#)," 06 February 2001.
- 1993 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI):
- 1994 Generic Syntax," RFC 3986, MIT/LCS, Day Software, Adobe Systems, January
- 1995 2005.
- 1996 **[XPath]** W3C Recommendation, "[XML Path Language](#)", 16 November 1999
- 1997
- 1998 The following are non-normative references included for background and related material:
- 1999 **[WS-SECURITY]** "[Web Services Security Language](#)", IBM, Microsoft, VeriSign, April 2002.
- 2000 "[WS-Security Addendum](#)", IBM, Microsoft, VeriSign, August 2002.
- 2001 "[WS-Security XML Tokens](#)", IBM, Microsoft, VeriSign, August 2002.
- 2002 **[XMLC14N]** W3C Recommendation, "[Canonical XML Version 1.0](#)," 15 March 2001.
- 2003 **[EXCC14N]** W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8 July
- 2004 2002.
- 2005 **[XMLENC]** W3C Working Draft, "[XML Encryption Syntax and Processing](#)," 04 March 2002.
- 2006 W3C Recommendation, "Decryption Transform for XML Signature", 10 December 2002.
- 2007 **[XML-ns]** W3C Recommendation, "[Namespaces in XML](#)," 14 January 1999.
- 2008 **[XMLSCHEMA]** W3C Recommendation, "[XML Schema Part 1: Structures](#)," 2 May 2001.
- 2009 W3C Recommendation, "[XML Schema Part 2: Datatypes](#)," 2 May 2001.
- 2010 **[XMLSIG]** D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-*
- 2011 *Signature Syntax and Processing*, W3C Recommendation, 12 February 2002.
- 2012 **[X509]** S. Santesson, et al, "Internet X.509 Public Key Infrastructure Qualified Certificates
- 2013 Profile,"
- 2014 [http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-](http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I)
- 2015 [X.509-200003-I](http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.509-200003-I)
- 2016 **[WSS-SAML]** OASIS Working Draft 06, "Web Services Security SAML Token Profile", 21
- 2017 February 2003
- 2018 **[WSS-XrML]** OASIS Working Draft 03, "Web Services Security XrML Token Profile", 30
- 2019 January 2003
- 2020 **[WSS-X509]** OASIS, "Web Services Security X.509 Certificate Token Profile", 19 January
- 2021 2004, [http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf)
- 2022 [profile-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf)
- 2023 **[WSSKERBEROS]** OASIS Working Draft 03, "Web Services Security Kerberos Profile", 30 January
- 2024 2003
- 2025 **[WSSUSERNAME]** OASIS, "Web Services Security UsernameToken Profile" 19 January 2004,
- 2026 [http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf)
- 2027 [profile-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf)

2028	<b>[WSS-XCBF]</b>	OASIS Working Draft 1.1, "Web Services Security XCBF Token Profile", 30
2029		March 2003
2030	<b>[XMLID]</b>	W3C Recommendation, "xml:id Version 1.0", 9 September 2005.
2031	<b>[XPOINTER]</b>	"XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation",
2032		DeRose, Maler, Daniel, 11 September 2001.

---

2033 **# Conformance**

2034 The last numbered section in the specification must be the Conformance section. Conformance  
2035 Statements/Clauses go here.

2036

## A. Acknowledgements

2037 The following individuals have participated in the creation of this specification and are gratefully  
2038 acknowledged:

2039 **Participants:**

2040 **Current Contributors:**

Tom	Rutt	Fujitsu Limited
Jacques	Durand	Fujitsu Limited
Calvin	Powers	IBM
Kelvin	Lawrence	IBM
Michael	McIntosh	Individual
Thomas	Hardjono	M.I.T.
David	Turner	Microsoft Corporation
Anthony	Nadalin	Microsoft Corporation
Monica	Martin	Microsoft Corporation
Marc	Goodner	Microsoft Corporation
Peter	Davis	Neustar
Hal	Lockhart	Oracle Corporation
Rich	Levinson	Oracle Corporation
Anil	Saldhana	Red Hat
Martin	Raepple	SAP AG
Federico	Rossini	Telecom Italia S.p.a.
Carlo	Milono	TIBCO Software Inc.
Don	Adams	TIBCO Software Inc.
Jerry	Smith	US Department of Defense (DoD)

2041 **Previous Contributors:**

Michael	Hu	Actional
Maneesh	Sahu	Actional
Duane	Nickull	Adobe Systems
Gene	Thurston	AmberPoint
Frank	Siebenlist	Argonne National Laboratory
Pete	Dapkus	BEA
Hal	Lockhart	BEA Systems
Denis	Pilipchuk	BEA Systems
Corinna	Witt	BEA Systems
Steve	Anderson	BMC Software

Rich	Levinson	Computer Associates
Thomas	DeMartini	ContentGuard
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard
Xin	Wang	ContentGuard
Merlin	Hughes	Cybertrust
Dale	Moberg	Cyclone Commerce
Shawn	Sharp	Cyclone Commerce
Rich	Salz	Datapower
Ganesh	Vaideeswaran	Documentum
Sam	Wei	EMC
Tim	Moses	Entrust
Carolina	Canales-Valenzuela	Ericsson
Dana S.	Kaufman	Forum Systems
Toshihiro	Nishimura	Fujitsu
Tom	Rutt	Fujitsu
Kefeng	Chen	GeoTrust
Irving	Reid	Hewlett-Packard
Kojiro	Nakayama	Hitachi
Yutaka	Kudo	Hitachi
Jason	Rouault	HP
Paula	Austel	IBM
Derek	Fu	IBM
Maryann	Hondo	IBM
Kelvin	Lawrence	IBM
Michael	McIntosh	IBM
Anthony	Nadalín	IBM
Nataraj	Nagaratnam	IBM
Bruce	Rich	IBM
Ron	Williams	IBM
Bob	Blakley	IBM
Joel	Farrell	IBM
Satoshi	Hada	IBM
Hiroshi	Maruyama	IBM
David	Melgar	IBM
Kent	Tamura	IBM

Wayne	Vicknair	IBM
Don	Flinn	Individual
Phil	Griffin	Individual
Mark	Hayes	Individual
John	Hughes	Individual
Peter	Rostin	Individual
Davanum	Srinivas	Individual
Bob	Morgan	Individual/Internet
Kate	Cherry	Lockheed Martin
Bob	Atkinson	Microsof
Paul	Cotton	Microsoft
Vijay	Gajjala	Microsoft
Martin	Gudgin	Microsoft
Chris	Kaler	Microsoft
Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Giovanni	Della-Libera	Microsoft
Alan	Geller	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
Jeff	Hodges	Neustar
Frederick	Hirsch	Nokia
Senthil	Sengodan	Nokia
Abbie	Barbir	Nortel
Lloyd	Burch	Novell
Ed	Reed	Novell
Charles	Knouse	Oblix
Prateek	Mishra	Oracle
Vamsi	Motukuru	Oracle
Ramana	Turlapi	Oracle
Vipin	Samar	Oracle
Jerry	Schwarz	Oracle

Eric	Gravengaard	Reactivity
Andrew	Nash	Reactivity
Stuart	King	Reed Elsevier
Ben	Hammond	RSA Security
Rob	Philpott	RSA Security
Martijn	de Boer	SAP
Blake	Dournaee	Sarvega
Sundeeep	Peechu	Sarvega
Coumara	Radja	Sarvega
Pete	Wenzel	SeeBeyond
Jonathan	Tourzan	Sony
Yassir	Elley	Sun
Manveen	Kaur	Sun Microsystems
Ronald	Monzillo	Sun Microsystems
Jan	Alexander	Systinet
Michael	Nguyen	The IDA of Singapore
Don	Adams	TIBCO Software Inc.
Symon	Chang	TIBCO Software Inc.
John	Weiland	US Navy
Hans	Granqvist	VeriSign
Phillip	Hallam-Baker	VeriSign
Hemma	Prafullchandra	VeriSign
Morten	Jorgensen	Vordel

2042

## B. Revision History

2043

Revision	Date	Editor	Changes Made
WD01	17-January-2011	Carlo Milono	Corrected/added hyperlinks where missing; added Status section
WD02	8-February-2011	Carlo Milono	Added Related Work to reflect v1.1.1 of the specs; changed References for SOAP Message Security to reflect v1.1.1; Changed WD# to 2; Added Date; Moved Current Members to Previous and added new Current Members; saved document under wd02; entered the Revision History  Merged Old Current Contributors with Old Previous, created a New Current Contributors.
WD03	16-March-2011	David Turner	Corrected and updated links.

2044

2045

## C. Utility Elements and Attributes

2046  
2047  
2048  
2049

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

2050

### C.1 Identification Attribute

2051  
2052  
2053  
2054  
2055  
2056

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

2057

2058  
2059  
2060  
2061

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

2062

2063  
2064  
2065

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

2066

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

2067

2068

This section is non-normative.

2069

### C.2 Timestamp Elements

2070  
2071  
2072

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

2073

2074  
2075  
2076  
2077  
2078

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

2079

The following table provides an overview of these elements:

2080

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.

2081  
2082 A detailed description can be found in Section 10.  
2083  
2084 This section is non-normative.

### 2085 C.3 General Schema Types

2086 The schema for the utility aspects of this specification also defines some general purpose schema  
2087 elements. While these elements are defined in this schema for use with this specification, they are  
2088 general purpose definitions that may be used by other specifications as well.

2089  
2090 Specifically, the following schema elements are defined and can be re-used:  
2091

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

2092  
2093 This section is non-normative.

2094

## D. SecurityTokenReference Model

2095

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

2096

2097

2098

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

2099

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

2100

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

2101

2102

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

2103

2104

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

2105

2106

2107

2108

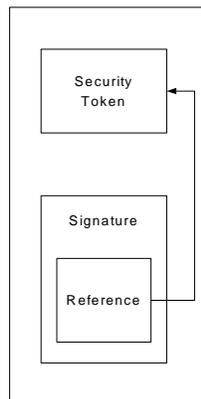
- The XML Signature references may include aspects (e.g. transforms) that may not apply to all references.

2109

2110

The following use cases drive the above motivations:

2111



2112

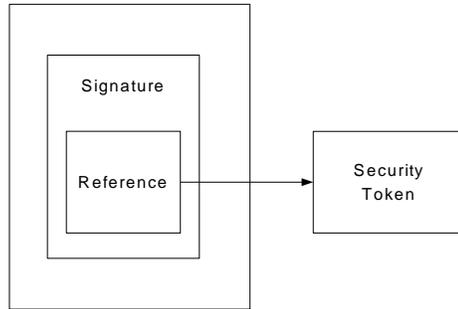
**Local Reference** – A security token, that is included in the message in the `<wsse:Security>` header, is associated with an XML Signature. The figure below illustrates this:

2113

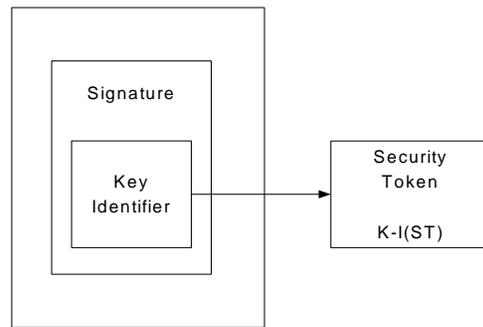
2114

2115

2116  
 2117 **Remote Reference** – A security token, that is not included in the message but may be available at a  
 2118 specific URI, is associated with an XML Signature. The figure below illustrates this:  
 2119

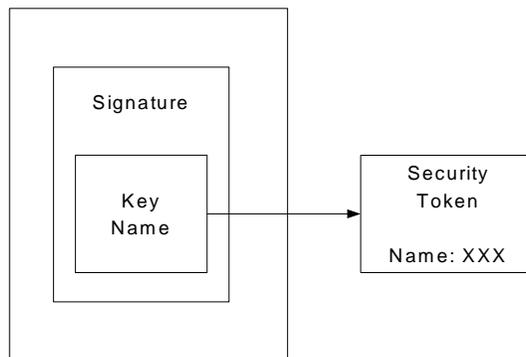


2120  
 2121 **Key Identifier** – A security token, which is associated with an XML Signature and identified using a

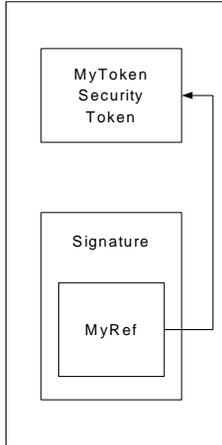


2122 known value that is the result of a well-known function of the security token (defined by the token format  
 2123 or profile). The figure below illustrates this where the token is located externally:  
 2124

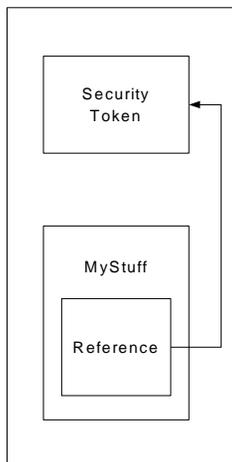
2125 **Key Name** – A security token is associated with an XML Signature and identified using a known value  
 2126 that represents a "name" assertion within the security token (defined by the token format or profile). The  
 2127 figure below illustrates this where the token is located externally:



2128  
 2129 **Format-Specific References** – A security token is associated with an XML Signature and identified using  
 2130 a mechanism specific to the token (rather than the general mechanisms described above). The figure  
 2131 below illustrates this:  
 2132



2133 **Non-Signature References** – A message may contain XML that does not represent an XML signature,  
 2134 but may reference a security token (which may or may not be included in the message). The figure below  
 2135 illustrates this:



2136  
 2137  
 2138 All conformant implementations must be able to process the `<wsse:SecurityTokenReference>`  
 2139 element. However, they are not required to support all of the different types of references.

2140  
 2141 The reference may include a `wsse11:TokenType` attribute which provides a "hint" for the type of desired  
 2142 token.

2143  
 2144 If multiple sub-elements are specified, together they describe the reference for the token.

2145 There are several challenges that implementations face when trying to interoperate:

2146 **ID References** – The underlying XML referencing mechanism using the XML base type of ID provides a  
 2147 simple straightforward XML element reference. However, because this is an XML type, it can be bound  
 2148 to *any* attribute. Consequently in order to process the IDs and references requires the recipient to  
 2149 *understand* the schema. This may be an expensive task and in the general case impossible as there is  
 2150 no way to know the "schema location" for a specific namespace URI.

2151  
 2152 **Ambiguity** – The primary goal of a reference is to uniquely identify the desired token. ID references are,  
 2153 by definition, unique by XML. However, other mechanisms such as "principal name" are not required to  
 2154 be unique and therefore such references may be unique.

2155 The XML Signature specification defines a `<ds:KeyInfo>` element which is used to provide information  
 2156 about the "key" used in the signature. For token references within signatures, it is recommended that the  
 2157 `<wsse:SecurityTokenReference>` be placed within the `<ds:KeyInfo>`. The XML Signature  
 2158 specification also defines mechanisms for referencing keys by identifier or passing specific keys. As a

2159 rule, the specific mechanisms defined in WSS: SOAP Message Security or its profiles are preferred over  
2160 the mechanisms in XML Signature.

2161 The following provides additional details on the specific reference mechanisms defined in WSS: SOAP  
2162 Message Security:

2163

2164 **Direct References** – The `<wsse:Reference>` element is used to provide a URI reference to the  
2165 security token. If only the fragment is specified, then it references the security token within the document  
2166 whose `wsu:Id` matches the fragment. For non-fragment URIs, the reference is to a [potentially external]  
2167 security token identified using a URI. There are no implied semantics around the processing of the URI.

2168

2169 **Key Identifiers** – The `<wsse:KeyIdentifier>` element is used to reference a security token by  
2170 specifying a known value (identifier) for the token, which is determined by applying a special *function* to  
2171 the security token (e.g. a hash of key fields). This approach is typically unique for the specific security  
2172 token but requires a profile or token-specific function to be specified. The `ValueType` attribute defines  
2173 the type of key identifier and, consequently, identifies the type of token referenced. The `EncodingType`  
2174 attribute specifies how the unique value (identifier) is encoded. For example, a hash value may be  
2175 encoded using base 64 encoding.

2176

2177 **Key Names** – The `<ds:KeyName>` element is used to reference a security token by specifying a specific  
2178 value that is used to *match* an identity assertion within the security token. This is a subset match and  
2179 may result in multiple security tokens that match the specified name. While XML Signature doesn't imply  
2180 formatting semantics, WSS: SOAP Message Security recommends that X.509 names be specified.

2181

2182 It is expected that, where appropriate, profiles define if and how the reference mechanisms map to the  
2183 specific token profile. Specifically, the profile should answer the following questions:

- 2184 • What types of references can be used?
- 2185 • How "Key Name" references map (if at all)?
- 2186 • How "Key Identifier" references map (if at all)?
- 2187 • Are there any additional profile or format-specific references?

2188

2189 This section is non-normative.

2190