Web Services Security: SOAP Message Security

Working Draft 17, Wednesday, 27 August 2003

Document identifier:

WSS: SOAP Message Security -17

Location:

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

Editors:

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

Contributors:

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

Deleted: 16
Deleted: Tuesday
Deleted: 26
Deleted: 6
Deleted: 16
Deleted: 6

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

Deleted: 16

Deleted: 6

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

Abstract:

This specification describes enhancements to SOAP messaging to provide message integrity, and single message authentication. 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 (e.g. support multiple security token formats). For example, a client might provide one format for proof of identity and provide another format for proof that they have a particular business certification.

Additionally, this specification describes how to encode binary security tokens, a framework for XML-based tokens, and 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 is an interim draft. Please send comments to the editors.

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

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 Security Services TC web page (http://www.oasis-open.org/who/intellectualproperty.shtml).

Deleted: 16

Deleted: 6

Table of Contents

Deleted: 3
Deleted: 6
Inserted: 3
Deleted: 3
Inserted: 3
Deleted: 6
Deleted: 3
Inserted: 3
Deleted: 6
Deleted: 3
Inserted: 3
Deleted: 6
Deleted: 3
Deleted: 8
Inserted: 3
Deleted: 3
Inserted: 3
Deleted: 8
Deleted: 3
Inserted: 3
Deleted: 8
Deleted: 3
Inserted: 3
Deleted: 9
Deleted: 3
Inserted: 3
Deleted: 10
Deleted: 3
Inserted: 3
Deleted: 10
Deleted: 3
Inserted: 3
Deleted: 10
Deleted: 3
Inserted: 3
Deleted: 11
Deleted: 3
Inserted: 3
Deleted: 11
Deleted: 3
Inserted: 3
Deleted: 13
Deleted: 3
Inserted: 3
Deleted: 13
Deleted: 3
Inserted: 3
Deleted: 13
Deleted: 3
Inserted: 3
Deleted: 15
Deleted: 3

Deleted: 3
Deleted: 26
Inserted: 3
Deleted: 3
Inserted: 3
Deleted: 28
Deleted: 3
Inserted: 3
Deleted: 28
Deleted: 3
Inserted: 3
Deleted: 30
Deleted: 3
Deleted: 30
Inserted: 3
Deleted: 3
Deleted: 31
Inserted: 3
Deleted: 3
Inserted: 3
Deleted: 32
Deleted: 3
Inserted: 3
Deleted: 32
Deleted: 3
Inserted: 3
Deleted: 32
Deleted: 3
Inserted: 3
Deleted: 33
Deleted: 3
Inserted: 3
Deleted: 34
Deleted: 3
Inserted: 3
Deleted: 36
Deleted: 3
Inserted: 3
Deleted: 39
Deleted: 3
Inserted: 3
Deleted: 40
Deleted: 3
Inserted: 3
Deleted: 42
Deleted: 3
Inserted: 3
Deleted: 43
Deleted: 3
Inserted: 3
Deleted: 44
Deleted: 3

1 Introduction

This specification proposes a standard set of SOAP 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 as the "Web Services Security Core Language" or "WSS-Core".

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 token 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 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:

- Multiple security token formats
- Multiple trust domains
- Multiple signature formats
- Multiple encryption technologies

End-to-end message content security and not just transport-level security

Deleted: 16

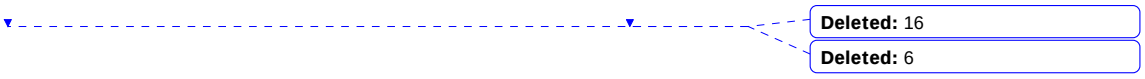
Deleted: 6

1.1.2 Non-Goals

The following topics are outside the scope of this document:

- Establishing a security context or authentication mechanisms.

- Key derivation.
- Advertisement and exchange of security policy.
- How trust is established or determined.



2 Notations and Terminology

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

2.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in 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 the notational convention of WSS: SOAP Message Security. Specifically, 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/>)

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

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

2.2 Namespaces

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

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

The above URIs contain versioning information as part of the URI. Any changes to this specification that cause different processing semantics must update the URI.

The following namespaces are used in this document:

Prefix	Namespace
S	http://www.w3.org/2002/12/soap-envelope
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#
wsse	http://schemas.xmlsoap.org/ws/2003/06/secext
wsu	http://schemas.xmlsoap.org/ws/2003/06/utility

Deleted: 16

Deleted: 6

2.3 Terminology

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

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

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

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

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

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

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

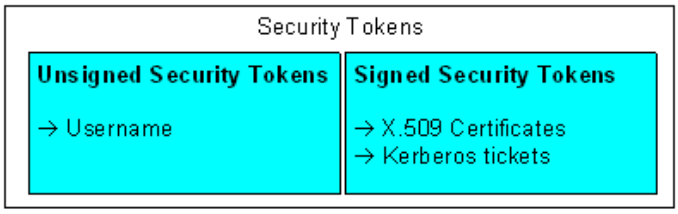
Message Confidentiality - *Message Confidentiality* is a property of the message and encryption is the mechanism by which this property of the message is provided.

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

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

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

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



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

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

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

Deleted: 16

Deleted: 6

3 Message Protection Mechanisms

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

3.1 Message Security Model

This document specifies an abstract *message security model* in terms of security tokens combined with digital signatures to protect and authenticate SOAP messages.
Security tokens assert claims and can be used to assert the binding between authentication secrets or keys and security identities. An authority can vouch for or endorse the claims in a security token by using its key to sign or encrypt (it is recommended to use a keyed encryption) the security token thereby enabling the authentication of the claims in the token. An X.509 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 sender of the containing message.
Signatures are used to verify message origin and integrity. Signatures are also used by message senders to demonstrate knowledge of the key 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 in conjunction with security tokens to ensure that modifications to messages detected. The integrity mechanisms are designed to support multiple signatures, potentially by multiple SOAP roles, and to be extensible to support additional signature formats.
Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of a SOAP message confidential. The encryption mechanisms are designed to support additional encryption processes and operations by multiple SOAP roles.
This document defines syntax and semantics of signatures within <wsse:Security> element.
This document does not specify any signature appearing outside of <wsse:Security> element.

Deleted: 16
Deleted: 6

3.3 Invalid or Missing Claims

The message recipient SHOULD reject a message with an invalid signature, a message that is missing necessary claims and a message whose claims have unacceptable values as such messages are unauthorized (or malformed) message.. This specification provides a flexible way for the message sender to make a claim about the security properties by associating zero or more security tokens with the message. An example of a security claim is the identity of the sender; the sender can claim that he is Bob, known as an employee of some company, and therefore he has the right to send the message.

3.4 Example

The following example illustrates the use of a custom security token and associated signature.. The token contains base64 encoded binary data which conveys a symmetric key to the recipient. The message sender uses the symmetric key with an HMAC signing algorithm to sign the message. The message receiver uses its knowledge of the shared secret to repeat the HMAC key calculation which it uses to validate the signature and in the process confirm that the message was authored by the claimed user identity.

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
                  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
(003)   <S:Header>
(004)      <wsse:Security
                xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
(005)         <xxx:CustomToken wsu:Id="MyID"
                               xmlns:xxx="http://fabrikam123/token">
(006)             FHUIORv...
(007)         </xxx:CustomToken>
(008)         <ds:Signature>
(009)            <ds:SignedInfo>
(010)               <ds:CanonicalizationMethod
                        Algorithm=
                          "http://www.w3.org/2001/10/xml-exc-c14n#"/>
(011)               <ds:SignatureMethod
                        Algorithm=
                        "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
(012)               <ds:Reference URI="#MsgBody">
(013)                  <ds:DigestMethod
                          Algorithm=
                        "http://www.w3.org/2000/09/xmldsig#sha1"/>
(014)                  <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
(015)               </ds:Reference>
(016)            </ds:SignedInfo>
(017)            <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
(018)            <ds:KeyInfo>
(019)                <wsse:SecurityTokenReference>
(020)                   <wsse:Reference URI="#MyID"/>
(021)                </wsse:SecurityTokenReference>
(022)            </ds:KeyInfo>
(023)         </ds:Signature>
(024)      </wsse:Security>
(025)   </S:Header>
(026)   <S:Body wsu:Id="MsgBody">
(027)     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
              QQQ
          </tru:StockSymbol>
```

Deleted: 16

Deleted: 6

```
(028)   </S:Body>
(029) </S:Envelope>
```

The first two lines start the SOAP envelope. Line (003) begins the headers that are associated with this SOAP message.
Line (004) starts the `<Security>` header defined in this specification. This header contains security information for an intended recipient. This element continues until line (024)
Lines (005) to (007) specify a custom token that is associated with the message. In this case, it uses an externally defined custom token format.
Lines (008) to (035) specify a digital signature. This signature ensures the integrity of the signed elements. The signature uses the XML Signature specification identified by the ds namespace declaration in Line (002). In this example, the signature is based on a key generated from the user's password; typically stronger signing mechanisms would be used (see the Extended Example later in this document).
Lines (009) to (016) describe what is being signed and the type of canonicalization being used.
Line (010) specifies how to canonicalize (normalize) the data that is being signed. Lines (012) to (015) select the elements that are signed and how to digest them. Specifically, line (012) indicates that the <S:Body> element is signed. In this example only the message body is signed; typically all critical elements of the message are included in the signature (see the Extended Example below).
Line (017) specifies the signature value of the canonicalized form of the data that is being signed as defined in the XML Signature specification.
Lines (018) to (022) provide a *hint* as to where to find the security token associated with this signature. Specifically, lines (019) to (021) indicate that the security token can be found at (pulled from) the specified URL.
Lines (026) to (028) contain the *body* (payload) of the SOAP message.

Deleted: 16

Deleted: 6

4 ID References

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

- Local ID attributes on XML Signature elements
- Local ID attributes on XML Encryption elements
- Global wsu:Id attributes (described below) on elements

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

4.1 Id Attribute

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 scope of 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 must be able to obtain the schemas where the identity or reference mechanisms are defined. In some circumstances, for example, intermediaries, this can be problematic and not desirable.

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.

This section 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 a particular attribute.

4.2 Id Schema

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

The syntax for this attribute is as follows:

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

The following describes the attribute illustrated above:

.../@wsu:Id

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

Deleted: 16

Deleted: 6

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

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

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

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

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

Deleted: 16

Deleted: 6

5 Security Header

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

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

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

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

The following illustrates the syntax of this header:

```
<S:Envelope>
    <S:Header>
            ...
        <wsse:Security S:role="..." S:mustUnderstand="...">
            ...
        </wsse:Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

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

/wsse:Security

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

/wsse:Security/@S:role

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

/wsse:Security/{any}

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

/wsse:Security/@{any}

Deleted: 16

Deleted: 6

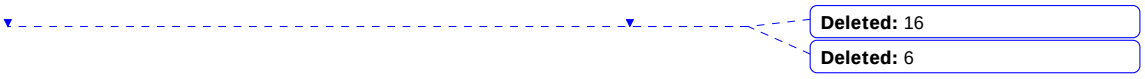
This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the header.

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

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

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

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



6 Security Tokens

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

6.1 Attaching Security Tokens

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

6.1.1 Processing Rules

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

6.1.2 Subject Confirmation

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

6.2 User Name Token

6.2.1 Usernames

The <wsse:UsernameToken> element is introduced as a way of providing a username. This element is optionally included in the <wsse:Security> header.
The following illustrates the syntax of this element:

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

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

/wsse:UsernameToken
: This element is used to represent a claimed identity.

/wsse:UsernameToken/@wsu:Id
: A string label for this security token.

/wsse:UsernameToken/Username
: This required element specifies the claimed identity.

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

Deleted: 16
Deleted: 6

/wsse:UsernameToken/{any}

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

/wsse:UsernameToken/@{any}

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

All compliant implementations MUST be able to process a `<wsse:UsernameToken>` element. The following illustrates the use of this:

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Zoe</wsse:Username>
            </wsse:UsernameToken>
        </wsse:Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

6.3 Binary Security Tokens

6.3.1 Attaching Security Tokens

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

6.3.2 Encoding Binary Security Tokens

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

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

The following is an overview of the syntax:

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

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

/wsse:BinarySecurityToken

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

/wsse:BinarySecurityToken/@wsu:Id

An optional string label for this security token.

/wsse:BinarySecurityToken/@ValueType

The `ValueType` attribute is used to indicate the "value space" of the encoded binary data (e.g. an X.509 certificate). The `ValueType` attribute allows a qualified name that defines the value type and space of the encoded binary data. This attribute is extensible

Deleted: 16

Deleted: 6

using XML namespaces. Subsequent specifications MUST define the ValueType value for the tokens that they define. The usage of ValueType is RECOMMENDED.

/wsse:BinarySecurityToken/@EncodingType

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

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

/wsse:BinarySecurityToken/@{any}

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

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

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

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

```
<wsse:BinarySecurityToken
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
        wsu:Id="myToken"
        ValueType="x:MyType" xmlns:x="http://www.fabrikam123.com/x"
        EncodingType="wsse:Base64Binary">
    MIIEZzCCA9CgAwIBAgIQEmtJZc0...
</wsse:BinarySecurityToken>
```

6.4 XML Tokens

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

6.4.1 Identifying and Referencing Security Tokens

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

Deleted: 16

Deleted: 6

7 Token References

This chapter discusses and defines mechanisms for referencing security tokens.

7.1 SecurityTokenReference Element

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

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

The following describes the elements defined above:

/wsse:SecurityTokenReference
: This element provides a reference to a security token.

/wsse:SecurityTokenReference/@wsu:Id
: A string label for this security token reference. This identifier names the reference. This attribute does not indicate the ID of what is being referenced, that SHALL be done using a fragment URI in a `<Reference>` element within the `<SecurityTokenReference>` element.

/wsse:SecurityTokenReference/@wsse:Usage
: This optional attribute is used to type the usage of the `<SecurityToken>`. Usages are specified using QNames and multiple usages MAY be specified using XML list semantics.

QName	Description
TBD	TBD

/wsse:SecurityTokenReference/{any}
: This is an extensibility mechanism to allow different (extensible) types of security references, based on a schema, to be passed.

/wsse:SecurityTokenReference/@{any}
: This is an extensibility mechanism to allow additional attributes, based on schemas, to be added to the header.

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

Deleted: 16

Deleted: 6

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

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

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

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

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

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

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

7.2 Direct References

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

The following illustrates the syntax of this element:

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

The following describes the elements defined above:

/wsse:SecurityTokenReference/Reference

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

/wsse:SecurityTokenReference/Reference/@URI

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

/wsse:SecurityTokenReference/Reference/@ValueType

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

Deleted: 16

Deleted: 6

/wsse:SecurityTokenReference/Reference/{any}

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

/wsse:SecurityTokenReference/Reference/@{any}

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

The following illustrates the use of this element:

```
<wsse:SecurityTokenReference
         xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
   <wsse:Reference
            URI="http://www.fabrikam123.com/tokens/Zoe"/>
</wsse:SecurityTokenReference>
```

7.3 Key Identifiers

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

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

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

The following is an overview of the syntax:

```
<wsse:SecurityTokenReference>
   <wsse:KeyIdentifier wsu:Id="..."
                       ValueType="..."
                       EncodingType="...">
      ...
   </wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
```

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

/wsse:SecurityTokenReference /KeyIdentifier

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

/wsse:SecurityTokenReference/KeyIdentifier/@wsu:Id

An optional string label for this identifier.

/wsse:SecurityTokenReference/KeyIdentifier/@ValueType

The optional ValueType attribute is used to indicate the type of KeyIdentifier being used. Each token profile specifies the KeyIdentifier types that may be used to refer to tokens of that type. It also specifies the critical semantics of the identifier, such as whether the KeyIdentifier is unique to the key or the token. Any value specified for binary security tokens, or any XML token element QName can be specified here. If no value is specified then the key identifier will be interpreted in an application-specific manner.

/wsse:SecurityTokenReference/KeyIdentifier/@EncodingType

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

Formatted: ElementDesc, Line spacing: single

Deleted: Profiles must define what value is implied if a specific value is not specified

Deleted: .

Deleted: 16

Deleted: 6

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

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

7.4 Embedded References

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

```
<wsse:SecurityTokenReference>
   <wsse:Embedded wsu:Id="...">
      ...
   </wsse:Embedded>
</wsse:SecurityTokenReference>
```

The following describes the attributes and elements listed in the example above:
/wsse:SecurityTokenReference /Embedded
This element is used to embed a token directly within a reference (that is, to create a *local* or *literal* reference).
/wsse:SecurityTokenReference/Embedded/@wsu:Id
An optional string label for this element. This allows this embedded token to be referenced by a signature or encryption.
/wsse:SecurityTokenReference/Embedded/{any}
This is an extensibility mechanism to allow any security token, based on schemas, to be embedded.
/wsse:SecurityTokenReference/Embedded/@{any}
This is an extensibility mechanism to allow additional attributes, based on schemas, to be added.
The following example illustrates embedding a SAML assertion:

```
<S:Envelope>
    <S:Header>
        <wsse:Security>
            ...
            <wsse:SecurityTokenReference>
                <wsse:Embedded wsu:Id="tok1">
                    <saml:Assertion xmlns:saml="...">
                        ...
                    </saml:Assertion xmlns:saml="...">
                </wsse:Embedded>
            </wsse:SecurityTokenReference>
            ...
        <wsse:Security>
    </S:Header>
    ...
</S:Body>
```

Deleted: 16
Deleted: 6

7.5 ds:KeyInfo

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

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

7.6 Key Names

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

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

Deleted: 16
Deleted: 6

8 Signatures

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

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

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

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

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

8.1 Algorithms

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

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

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

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

Finally, if a sender wishes to sign a message before encryption, they should alter the order of the signature and encryption elements inside of the `<wsse:Security>` header.

Deleted: 16

Deleted: 6

8.2 Signing Messages

The <wsse:Security> header block MAY be used to carry a signature compliant with the XML Signature specification within a SOAP Envelope for the purpose of signing one or more elements in the SOAP Envelope. Multiple signature entries MAY be added into a single SOAP Envelope within one <wsse:Security> header block. Senders SHOULD take care to sign all important elements of the message, but care MUST be taken in creating a signing policy that requires signing of parts of the message that might legitimately be altered in transit.

SOAP applications MUST satisfy the following conditions:

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

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

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

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

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

8.3 Signing Tokens

It is often desirable to sign security tokens that are included in a message or even external to the message. The XML Signature specification provides several common ways for referencing information to be signed such as URIs, IDs, and XPath, but some token formats may not allow tokens to be referenced using URIs or IDs and XPaths may be undesirable in some situations. This specification allows different tokens to have their own unique reference mechanisms which are specified in their profile as extensions to the <SecurityTokenReference> element. This element provides a uniform referencing mechanism that is guaranteed to work with all token formats. Consequently, this specification defines a new reference option for XML Signature: the STR Dereference Transform.

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

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

Deleted: 16

Deleted: 6

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

```
...
<wsse:SecurityTokenReference wsu:Id="Str1">
     ...
</wsse:SecurityTokenReference>
...
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
     <SignedInfo>
      ...
       <Reference URI="#Str1">
         <Transforms>
           <ds:Transform
                Algorithm="http://schemas.xmlsoap.org/2003/06/STR-
Transform">
                <ds:CanonicalizationMethod
                      Algorithm="http://www.w3.org/TR/2001/REC-xml-
c14n-20010315" />
           </ds:Transform>
         <DigestMethod Algorithm=
                           "http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>...</DigestValue>
       </Reference>
     </SignedInfo>
     <SignatureValue></SignatureValue>
</Signature>
...
```

The following is a detailed specification of the transformation.
The algorithm is identified by the URI: http://schemas.xmlsoap.org/2003/06/STR-Transform
Transform Input:

- The input is a node set. If the input is an octet stream, then it is automatically parsed; cf. dsig.

Transform Output:
The output is an octet steam.
Syntax:
The transform takes a single mandatory parameter, a ds:CanonicalizationMethod, which is used to serialize the input node set. Note, however, that the output may not be strictly in canonical form, per the canonicalization algorithm; however, the output is canonical, in the sense that it is unambiguous.
Processing Rules:

- Let N be the input node set.
- Let R be the set of all wsse:SecurityTokenReference elements in N.
- For each Ri in R, let Di be the result of dereferencing Ri.
  - If Di cannot be determined, then the transform MUST signal a failure.
  - If Di is an XML security token (e.g., a SAML assertion or a wsse:BinarySecurityToken element), then let Ri' be Di.
  - Otherwise, Di is a raw binary security token; i.e., an octet stream. In this case, let Ri' be a node set consisting of a wsse:BinarySecurityToken element, utilizing the same namespace prefix as the wsse:SecurityTokenReference element Ri, with no EncodingType attribute, a ValueType attribute identifying the content of the security token, and text content consisting of the binary-encoded security token, with no whitespace. The ValueType QName MUST use the same namespace prefix as the BinarySecurityToken element if the QName has the same

Deleted: XML security token

Deleted: binary security token

Formatted: ElementDesc Char, Font: (Default) Helvetica, Font color: Auto

Deleted: 16

Deleted: 6

namespace URI. Otherwise, it MUST use the namespace prefix x, or else the prefix y if Ri uses x. If no appropriate ValueType QName is known, then the transform MUST signal a failure.

Formatted: ElementDesc Char

Deleted: prefix x

- Finally, employ the canonicalization method specified as a parameter to the transform to serialize N to produce the octet stream output of this transform; but, in place of any dereferenced wsse:SecurityTokenReference element Ri and its descendants, process the dereferenced node set Ri' instead. During this step, canonicalization of the replacement node-set MUST be augmented as follows:

Notes:

- A namespace declaration xmlns="" MUST be emitted with every apex element that has no namespace node declaring a value for the default namespace; cf. XML Decryption Transform.
- If the canonicalization algorithm is inclusive XML canonicalization and a node-set is replacing an element from N whose parent element is not in N, then its apex elements MUST inherit attributes associated with the XML namespace from the parent element., such as xml:base, xml:lang and xml:space.

8.4 Signature Validation

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

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

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

8.5 Example

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

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
   <S:Header>
      <wsse:Security>
         <wsse:BinarySecurityToken
                     ValueType="wsse:X509v3"
                     EncodingType="wsse:Base64Binary"
                     wsu:Id="X509Token">
                  MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
         </wsse:BinarySecurityToken>
         <ds:Signature>
            <ds:SignedInfo>
               <ds:CanonicalizationMethod Algorithm=
                     "http://www.w3.org/2001/10/xml-exc-c14n#"/>
```

Deleted: 16

Deleted: 6

```
                <ds:SignatureMethod Algorithm=
                      "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                <ds:Reference URI="#myBody">
                   <ds:Transforms>
                      <ds:Transform Algorithm=
                            "http://www.w3.org/2001/10/xml-exc-c14n#"/>
                   </ds:Transforms>
                   <ds:DigestMethod Algorithm=
                        "http://www.w3.org/2000/09/xmldsig#sha1"/>
                   <ds:DigestValue>EULddytSo1...</ds:DigestValue>
                </ds:Reference>
             </ds:SignedInfo>
             <ds:SignatureValue>
               BL8jdfToEb1l/vXcMZNNjPOV...
             </ds:SignatureValue>
             <ds:KeyInfo>
                 <wsse:SecurityTokenReference>
                     <wsse:Reference URI="#X509Token"/>
                 </wsse:SecurityTokenReference>
             </ds:KeyInfo>
          </ds:Signature>
       </wsse:Security>
    </S:Header>
    <S:Body wsu:Id="myBody">
       <tru:StockSymbol xmlns:tru="http://www.fabrikam123.com/payloads">
         QQQ
       </tru:StockSymbol>
    </S:Body>
</S:Envelope>
```

Deleted: 16

Deleted: 6

9 Encryption

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

9.1 xenc:ReferenceList

The `<xenc:ReferenceList>` element from XML Encryption MAY be used to create a manifest of encrypted portion(s), which are expressed as `<xenc:EncryptedData>` elements within the envelope. An element or element content to be encrypted by this encryption step MUST be replaced by a corresponding `<xenc:EncryptedData>` according to XML Encryption. All the `<xenc:EncryptedData>` elements created by this encryption step SHOULD be listed in `<xenc:DataReference>` elements inside one or more `<xenc:ReferenceList>` element.
Although in XML Encryption, `<xenc:ReferenceList>` was originally designed to be used within an `<xenc:EncryptedKey>` element (which implies that all the referenced `<xenc:EncryptedData>` elements are encrypted by the same key), this specification allows that `<xenc:EncryptedData>` elements referenced by the same `<xenc:ReferenceList>` MAY be encrypted by different keys. Each encryption key can be specified in `<ds:KeyInfo>` within individual `<xenc:EncryptedData>`.
A typical situation where the `<xenc:ReferenceList>` sub-element is useful is that the sender and the recipient use a shared secret key. The following illustrates the use of this sub-element:

```
<S:Envelope
   xmlns:S="http://www.w3.org/2001/12/soap-envelope"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
        <wsse:Security>
            <xenc:ReferenceList>
                <xenc:DataReference URI="#bodyID"/>
            </xenc:ReferenceList>
        </wsse:Security>
    </S:Header>
    <S:Body>
        <xenc:EncryptedData Id="bodyID">
          <ds:KeyInfo>
```

Deleted: 16
Deleted: 6

```
            <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
          </ds:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
          </xenc:CipherData>
        </xenc:EncryptedData>
    </S:Body>
</S:Envelope>
```

9.2 xenc:EncryptedKey

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

```
<S:Envelope
   xmlns:S="http://www.w3.org/2001/12/soap-envelope"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
        <wsse:Security>
              <xenc:EncryptedKey>
                ...
                <ds:KeyInfo>
                  <wsse:SecurityTokenReference>
                    <ds:X509IssuerSerial>
                      <ds:X509IssuerName>
                        DC=ACMECorp, DC=com
                       </ds:X509IssuerName>
<ds:X509SerialNumber>12345678</ds:X509SerialNumber>
                    </ds:X509IssuerSerial>
                  </wsse:SecurityTokenReference>
                </ds:KeyInfo>
                ...
              </xenc:EncryptedKey>
  ...
        </wsse:Security>
    </S:Header>
    <S:Body>
        <xenc:EncryptedData Id="bodyID">
            <xenc:CipherData>
              <xenc:CipherValue>...</xenc:CipherValue>
            </xenc:CipherData>
        </xenc:EncryptedData>
    </S:Body>
</S:Envelope>
```

Deleted: 16

Deleted: 6

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

9.3 Processing Rules

Encrypted parts or using one of the sub-elements defined above MUST be in compliance with the XML Encryption specification. An encrypted SOAP envelope MUST still be a valid SOAP envelope. The message creator MUST NOT encrypt the <S:Envelope>, <S:Header>, or <S:Body> elements but MAY encrypt child elements of either the <S:Header> and <S:Body> elements. Multiple steps of encryption MAY be added into a single <Security> header block if they are targeted for the same recipient.
When an element or element content inside a SOAP envelope (e.g. the contents of the <S:Body> element) is to be encrypted, it MUST be replaced by an <xenc:EncryptedData>, according to XML Encryption and it SHOULD be referenced from the <xenc:ReferenceList> element created by this encryption step.

9.3.1 Encryption

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

- Create a new SOAP envelope.
- Create a <Security> header
- Create an <xenc:ReferenceList> sub-element, an <xenc:EncryptedKey> sub-element, or an <xenc:EncryptedData> sub-element in the <Security> header block (note that if the SOAP "role" and "mustUnderstand" attributes are different, then a new header block may be necessary), depending on the type of encryption.
- Locate data items to be encrypted, i.e., XML elements, element contents within the target SOAP envelope.
- Encrypt the data items as follows: For each XML element or element content within the target SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification. Each selected original element or element content MUST be removed and replaced by the resulting <xenc:EncryptedData> element.
- The optional <ds:KeyInfo> element in the <xenc:EncryptedData> element MAY reference another <ds:KeyInfo> element. Note that if the encryption is based on an attached security token, then a <SecurityTokenReference> element SHOULD be added to the <ds:KeyInfo> element to facilitate locating it.
- Create an <xenc:DataReference> element referencing the generated <xenc:EncryptedData> elements. Add the created <xenc:DataReference> element to the <xenc:ReferenceList>.

9.3.2 Decryption

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

- Identify any decryption keys that are in the recipient's possession, then identifying any message elements that it is able to decrypt.
- Locate the <xenc:EncryptedData> items to be decrypted (possibly using the <xenc:ReferenceList>).

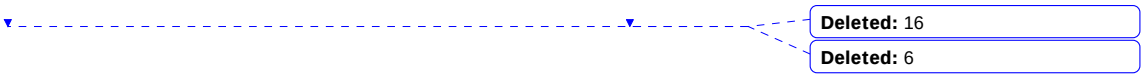
Deleted: 16
Deleted: 6

- Decrypt them as follows: For each element in the target SOAP envelope, decrypt it according to the processing rules of the XML Encryption specification and the processing rules listed above.
- If the decryption fails for some reason, applications MAY report the failure to the sender using the fault code defined in Section 12 Error Handling.

Parts of a SOAP message may be encrypted in such a way that they can be decrypted by an intermediary that is targeted by one of the SOAP headers. Consequently, the exact behavior of intermediaries with respect to encrypted data is undefined and requires an out-of-band agreement.

9.4 Decryption Transformation

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



10 Security Timestamps

It is often important for the recipient to be able to determine the *freshness* of security semantics. In some cases, security semantics may be so *stale* that the recipient may decide to ignore it.
This specification does not provide a mechanism for synchronizing time. The assumption is that time is trusted or additional mechanisms, not described here, are employed to prevent replay.
This specification defines and illustrates time references in terms of the *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this type. It is further RECOMMENDED that all references be in UTC time. Implementations MUST NOT generate time instants that specify leap seconds. If, however, other time types are used, then the *ValueType* attribute (described below) MUST be specified to indicate the data type of the time format. Requestors and receivers SHOULD NOT rely on other applications supporting time resolution finer than milliseconds.
The `<wsu:Timestamp>` element provides a mechanism for expressing the creation and expiration times of the security semantics in a message.
All times SHOULD be in UTC format as specified by the XML Schema type (dateTime). It should be noted that times support time precision as defined in the XML Schema specification.
The `<wsu:Timestamp>` element is specified as a child of the `<wsse:Security>` header and may only be present at most once per header (that is, per SOAP role).
The ordering within the element is as illustrated below. The ordering of elements in the <wsu:Timestamp> header is fixed and MUST be preserved by intermediaries.
To preserve overall integrity of each `<wsu:Timestamp>` element, it is strongly RECOMMENDED that each SOAP role only create or update the appropriate `<wsu:Timestamp>` element destined to itself (that is, a `<wsse:Security>` header whose actor/role is itself) and no other `<wsu:Timestamp>` element.
The schema outline for the `<wsu:Timestamp>` element is as follows:

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

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

/wsu:Timestamp
: This is the header for indicating message timestamps.

/wsu:Timestamp/wsu:Created
: This represents the creation time of the security semantics. This element is optional, but can only be specified once in a `Timestamp` element. Within the SOAP processing model, creation is the instant that the infoset is serialized for transmission. The creation time of the message SHOULD NOT differ substantially from its transmission time. The difference in time should be minimized.

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

/wsu:Timestamp/wsu:Expires
: This represents the expiration of the security semantics. This is optional, but can appear at most once in a `Timestamp` element. Upon expiration, the requestor asserts that its

Deleted: 16
Deleted: 6

security semantics are no longer valid. It is strongly RECOMMENDED that recipients (anyone who processes this message) discard (ignore) any message whose security semantics have passed their expiration. A Fault code (wsu:MessageExpired) is provided if the recipient wants to inform the requestor that its security semantics were expired. A service MAY issue a Fault indicating the security semantics have expired.

/wsu:Timestamp/wsu:Expires/@ValueType

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

/wsu:Timestamp/{any}

This is an extensibility mechanism to allow additional elements to be added to the element.

/wsu:Timestamp/@wsu:Id

This optional attribute specifies an XML Schema ID that can be used to reference this element (the timestamp). This is used, for example, to reference the timestamp in a XML Signature.

/wsu:Timestamp/@{any}

This is an extensibility mechanism to allow additional attributes to be added to the element.

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

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

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
  <S:Header>
    <wsse:Security>
      <wsu:Timestamp wsu:Id="timestamp">
         <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
         <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
      </wsu:Timestamp>
      ...
    </wsse:Security>
    ...
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

Deleted: 16

Deleted: 6

11 Extended Example

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

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
(003)   <S:Header>
(004)      <wsse:Security>
(005)         <wsu:Timestamp>
(006)           <wsu:Created
(007)                   wsu:Id="T0">2001-09-13T08:42:00Z</wsu:Created>
(008)         </wsu:Timestamp>
(009)
(010)         <wsse:BinarySecurityToken
                     ValueType="wsse:X509v3"
                     wsu:Id="X509Token"
                     EncodingType="wsse:Base64Binary">
(011)         MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
(012)         </wsse:BinarySecurityToken>
(013)         <xenc:EncryptedKey>
(014)             <xenc:EncryptionMethod Algorithm=
                        "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
(015)             <ds:KeyInfo>
(016)               <wsse:KeyIdentifier
EncodingType="wsse:Base64Binary"
                     ValueType="wsse:X509v3">MIGfMa0GCSq...
(017)               </wsse:KeyIdentifier>
(018)             </ds:KeyInfo>
(019)             <xenc:CipherData>
(020)               <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(021)               </xenc:CipherValue>
(022)             </xenc:CipherData>
(023)             <xenc:ReferenceList>
(024)                <xenc:DataReference URI="#enc1"/>
(025)             </xenc:ReferenceList>
(026)         </xenc:EncryptedKey>
(027)         <ds:Signature>
(028)           <ds:SignedInfo>
(029)             <ds:CanonicalizationMethod
                     Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(030)               <ds:SignatureMethod
                  Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(031)               <ds:Reference URI="#T0">
(032)                 <ds:Transforms>
(033)                   <ds:Transform
                     Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(034)                 </ds:Transforms>
```

Deleted: 5
Deleted:
Deleted: (016)
Deleted: 017
Deleted: 8
Deleted: 19
Deleted: 0
Deleted: 1
Deleted: 2
Deleted: 3
Deleted: 4
Deleted: 5
Deleted: 6
Deleted: 7
Deleted: 8
Deleted: 29
Deleted: 16
Deleted: 6
Deleted: 039
Deleted: 031
Deleted: 032
Deleted: 033

```
(035)                    <ds:DigestMethod
                       Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(036)                    <ds:DigestValue>LyLsF094hPi4wPU...
(037)                    </ds:DigestValue>
(038)                  </ds:Reference>
(039)                  <ds:Reference URI="#body">
(040)                    <ds:Transforms>
(041)                      <ds:Transform
                       Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(042)                    </ds:Transforms>
(043)                    <ds:DigestMethod
                       Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(044)                    <ds:DigestValue>LyLsF094hPi4wPU...
(045)                    </ds:DigestValue>
(046)                  </ds:Reference>
(047)                </ds:SignedInfo>
(048)                <ds:SignatureValue>
(049)                    Hp1ZkmFZ/2kQLXDJbchm5gK...
(050)                </ds:SignatureValue>
(051)                <ds:KeyInfo>
(052)                  <wsse:SecurityTokenReference>
(053)                    <wsse:Reference URI="#X509Token"/>
(054)                  </wsse:SecurityTokenReference>
(055)                </ds:KeyInfo>
(056)              </ds:Signature>
(057)            </wsse:Security>
(058)          </S:Header>
(059)          <S:Body wsu:Id="body">
(060)            <xenc:EncryptedData
                       Type="http://www.w3.org/2001/04/xmlenc#Element"
                       wsu:Id="enc1">
(061)              <xenc:EncryptionMethod
                     Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/>
(062)              <xenc:CipherData>
(063)                <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(064)                </xenc:CipherValue>
(065)              </xenc:CipherData>
(066)            </xenc:EncryptedData>
(067)          </S:Body>
(068)        </S:Envelope>
```

Let's review some of the key sections of this example:
Lines (003)-(058) contain the SOAP message headers.
Lines (004)-(057) represent the `<wsse:Security>` header block. This contains the security-related information for the message.
Lines (005)-(008) specify the timestamp information. In this case it indicates the creation time of the security semantics.
Lines (010)-(012) specify a security token that is associated with the message. In this case, it specifies an X.509 certificate that is encoded as Base64. Line (011) specifies the actual Base64 encoding of the certificate.
Lines (013)-(026) specify the key that is used to encrypt the body of the message. Since this is a symmetric key, it is passed in an encrypted form. Line (014) defines the algorithm used to encrypt the key. Lines (015)-(018) specify the identifier of the key that was used to encrypt the symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines (023)-(025) identify the encryption block in the message that uses this symmetric key. In this case it is only used to encrypt the body (Id="enc1").

Deleted: 034
Deleted: 035
Deleted: 036
Deleted: 037
Deleted: 8
Deleted: 039
Deleted: 0
Deleted: 1
Deleted: 2
Deleted: 3
Deleted: 4
Deleted: 5
Deleted: 6
Deleted: 7)
Deleted: 8
Deleted: 49
Deleted: 0
Deleted: 1
Deleted: 2
Deleted: 3
Deleted: 4
Deleted: 5
Deleted: 6
Deleted: 7
Deleted: 8
Deleted: 59
Deleted: 0
Deleted: 1
Deleted: 2
Deleted: 3
Deleted: 4
Deleted: 5
Deleted: 6
Deleted: 7
Deleted: 7
Deleted: 6
Deleted: 5…017…name 018…021…022…024 … [3]
Deleted: 16…6 … [4]

Lines (027)-(056) specify the digital signature. In this example, the signature is based on the X.509 certificate. Lines (028)-(047) indicate what is being signed. Specifically, line (039) references the message body.
Lines (048)-(050) indicate the actual signature value – specified in Line (043).
Lines (052)-(054) indicate the key that was used for the signature. In this case, it is the X.509 certificate included in the message. Line (053) provides a URI link to the Lines (010)-(012).
The body of the message is represented by Lines (057)-(067).
Lines (060)-(066) represent the encrypted metadata and form of the body using XML Encryption. Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line (061) specifies the encryption algorithm – Triple-DES in this case. Lines (063)-(064) contain the actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the key as the key references this encryption – Line (024).

Deleted: 026
Deleted: 055
Deleted: 027
Deleted: 6
Deleted: Line (039) references the creation timestamp and
Deleted: 8
Deleted: 7
Deleted: 49
Deleted: 2
Deleted: 051
Deleted: 053
Deleted: 052
Deleted: 056
Deleted: 066
Deleted: 059
Deleted: 065
Deleted: 0
Deleted: 062
Deleted: 063
Deleted: 023

Deleted: 16
Deleted: 6

12 Error Handling

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

- Invalid or unsupported type of security token, signing, or encryption
- Invalid or unauthenticated or unauthenticatable security token
- Invalid signature
- Decryption failure
- Referenced security token is unavailable
- Unsupported namespace

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

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

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

The "failure" class of errors are:

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

Deleted: 16

Deleted: 6

13 Security Considerations

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

- Timestamp
- Sequence Number
- Expirations
- Message Correlation

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

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

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

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

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

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

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

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

Deleted: 16

Deleted: 6

<SecurityToken> elements that it is confirming and that are not signed by their issuing authority.

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

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

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

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

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

When a password is encrypted in addition to the normal threats against any encryption, two password-specific threats must be considered: replay and guessing. If an attacker can impersonate a user by replaying an encrypted or hashed password, then learning the actual password is not necessary. One method of preventing replay is to use a nonce as mentioned previously. Generally it is also necessary to use a timestamp to put a ceiling on the number of previous nonces that must be stored. However, in order to be effective the nonce and timestamp must be signed. If the signature is also over the password itself, prior to encryption, then it would be a simple matter to used the signature to perform an offline guessing attack against the password. This threat can be countered in any of several ways including: don't include the password under the signature (the password will be verified later) or sign the encrypted password.

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

Deleted: 16

Deleted: 6

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 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 followthe 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 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.

Deleted: implementtaionsl

Deleted: 16

Deleted: 6

15 Privacy Considerations

If messages contain data that is sensitive or personal in nature or for any reason should not be visible to parties other than the sender and authorized recipients, the use of encryption, as described in this specification, is strongly RECOMMENDED.
This specification DOES NOT define mechanisms for making privacy statements or requirements.

Deleted: 16
Deleted: 6

16 References

[DIGSIG] Informational RFC 2828, "Internet Security Glossary," May 2000.

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

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

[SHA-1] FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce / National Institute of Standards and Technology. http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt

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

[SOAP12] W3C Working Draft**, "**SOAP Version 1.2 Part 1: Messaging Framework", 26 June 2002

[SOAP-SEC] W3C Note, "SOAP Security Extensions: Digital Signature," 06 February 2001.

[URI] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[WS-Security] "Web Services Security Language", IBM, Microsoft, VeriSign, April 2002.
"WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002.
"WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002.

[XML-C14N] W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001

[EXC-C14N] W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8 July 2002.

[XML-Encrypt] W3C Working Draft, "XML Encryption Syntax and Processing," 04 March 2002

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

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

[XML-Schema] W3C Recommendation, "XML Schema Part 1: Structures,"2 May 2001.
W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.

[XML Signature] W3C Recommendation, "XML Signature Syntax and Processing," 12 February 2002.

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

[XPath] W3C Recommendation, "XML Path Language", 16 November 1999

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

Deleted: 16

Deleted: 6

1574 1575	**[WSS-XrML]**	OASIS Working Draft 03, "Web Services Security XrML Token Profile", 30 January 2003
1576 1577	**[WSS-X509]**	OASIS Working Draft 03, "Web Services Security X509 Profile", 30 January 2003
1578 1579	**[WSS-Kerberos]**	OASIS Working Draft 03, "Web Services Security Kerberos Profile", 30 January 2003
1580 1581	**[WSS-Username]**	OASIS Working Draft 02, "Web Services Security UsernameToken Profile", 23 February 2003
1582 1583	**[WSS-XCBF]**	OASIS Working Draft 1.1, "Web Services Security XCBF Token Profile", 30 March 2003
1584 1585	**[XPointer]**	"XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation", DeRose, Maler, Daniel, 11 September 2001.

Deleted: 16

Deleted: 6

Appendix A: Utility Elements and Attributes

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.

A.1. Identification Attribute

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

A.2. Timestamp Elements

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.
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.
The following table provides an overview of these elements:

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

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

Deleted: ,
Deleted: , and receipt
Deleted: s
Deleted: <wsu:Received> ... [5]
Deleted: Message
Deleted: 16
Deleted: 6

A.3. General Schema Types

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

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

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

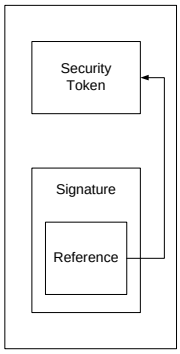
Deleted: 16

Deleted: 6

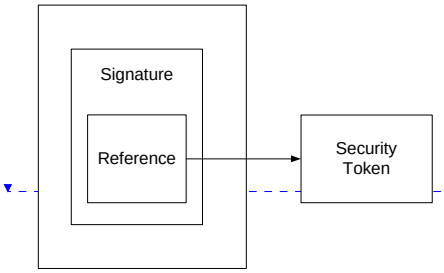
Appendix B: SecurityTokenReference Model

This appendix provides a non-normative overview of the usage and processing models for the <wsse:SecurityTokenReference> element.
There are several motivations for introducing the <wsse:SecurityTokenReference> element:
The XML Signature reference mechanisms are focused on "key" references rather than general token references.
The XML Signature reference mechanisms utilize a fairly closed schema which limits the extensibility that can be applied.
There are additional types of general reference mechanisms that are needed, but are not covered by XML Signature.
There are scenarios where a reference may occur outside of an XML Signature and the XML Signature schema is not appropriate or desired.
The XML Signature references may include aspects (e.g. transforms) that may not apply to all references.

The following use cases drive the above motivations:
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:

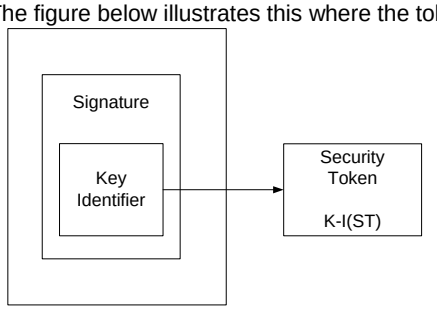


Remote Reference – A security token, that is not included in the message but may be available at a specific URI, is associated with an XML Signature. The figure below illustrates this:

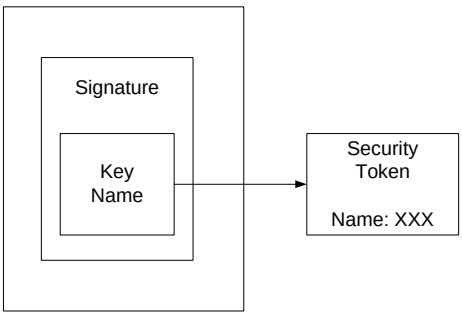


Deleted: 16
Deleted: 6

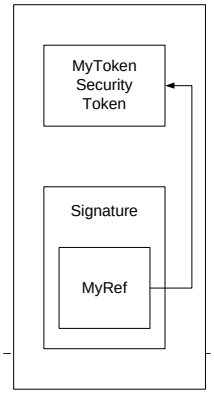
Key Identifier – A security token, which is associated with an XML Signature and identified using a known value that is the result of a well-known function of the security token (defined by the token format or profile). The figure below illustrates this where the token is located externally:



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



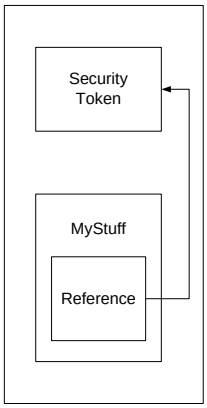
Format-Specific References – A security token is associated with an XML Signature and identified using a mechanism specific to the token (rather than the general mechanisms described above). The figure below illustrates this:



Deleted: 16

Deleted: 6

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



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

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

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

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

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

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

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

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

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

Key Identifiers – The `<wsse:KeyIdentifier>` element is used to reference a security token by specifying a known value (identifier) for the token, which is determined by applying a special

Deleted: 16

Deleted: 6

function to the security token (e.g. a hash of key fields). This approach is typically unique for the specific security token but requires a profile or token-specific function to be specified. The *ValueType* attribute defines the type of key identifier and, consequently, identifies the type of token referenced. The *EncodingType* attribute specifies how the unique value (identifier) is encoded. For example, a hash value may be encoded using base 64 encoding (the default).

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

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

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

Deleted: 16

Deleted: 6

Appendix C: Revision History

Rev	Date	What
01	20-Sep-02	Initial draft based on input documents and editorial review
02	24-Oct-02	Update with initial comments (technical and grammatical)
03	03-Nov-02	Feedback updates
04	17-Nov-02	Feedback updates
05	02-Dec-02	Feedback updates
06	08-Dec-02	Feedback updates
07	11-Dec-02	Updates from F2F
08	12-Dec-02	Updates from F2F
14	03-Jun-03	Completed these pending issues - 62, 69, 70, 72, 74, 84, 90, 94, 95, 96, 97, 98, 99, 101, 102, 103, 106, 107, 108, 110, 111
15	18-Jul-03	Completed these pending issues – 78, 82, 104, 105, 109, 111, 113
16	26-Aug-03	Completed these pending issues - 99, 128, 130, 132, 134

Deleted: 16

Deleted: 6

Appendix D: Notices

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

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

Deleted: 16

Deleted: 6

Page 1: [1] Deleted	**Anthony Nadalin**	**8/27/2003 8:54 PM**

16

Page 1: [1] Deleted	**Anthony Nadalin**	**8/27/2003 8:54 PM**

6

Page 1: [2] Deleted	**Anthony Nadalin**	**8/27/2003 8:54 PM**

16

Page 1: [2] Deleted	**Anthony Nadalin**	**8/27/2003 8:54 PM**

6

Page 37: [3] Deleted	**Anthony Nadalin**	**8/27/2003 8:42 PM**

5

Page 37: [3] Deleted	**Anthony Nadalin**	**8/27/2003 8:46 PM**

017

Page 37: [3] Deleted	**Anthony Nadalin**	**8/27/2003 8:19 PM**

name

Page 37: [3] Deleted	**Anthony Nadalin**	**8/27/2003 8:47 PM**

018

Page 37: [3] Deleted	**Anthony Nadalin**	**8/27/2003 8:47 PM**

021

Page 37: [3] Deleted	**Anthony Nadalin**	**8/27/2003 8:47 PM**

022

Page 37: [3] Deleted	**Anthony Nadalin**	**8/27/2003 8:47 PM**

024

Page 1: [4] Deleted	**Anthony Nadalin**	**8/27/2003 8:54 PM**

16

Page 1: [4] Deleted	**Anthony Nadalin**	**8/27/2003 8:54 PM**

6

Page 46: [5] Deleted	**Chris Kaler**	**8/26/2003 4:59 PM**

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