Web Services Security Core Specification

Working Draft 08, 12 December 2002

Document identifier:

WSS-Core-08

Location:

TBD

Editors:

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

Contributors:

TBD – Revise this list to include WSS TC contributors

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

Abstract:

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

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

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

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).

Table of Contents

1 Introduction

This specification proposes a standard set of SOAP extensions that can be used when building secure Web services to implement message level 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 binding 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 and providing a security 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-level security and not just transport-level security

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 RFC2119.

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

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

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

@- Attribute; prefix for an attribute name

{any} - Wildcard

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

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

2.2 Namespaces

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

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

The following namespaces are used in this document:

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

wsu	http://schemas.xmlsoap.org/ws/2002/xx/utility

2.3 Terminology

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

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

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

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.

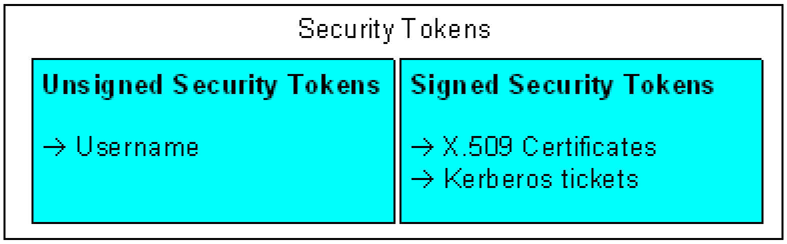
Message Confidentiality - *Message Confidentiality* is a property of the message and encryption is the service or mechanism by which this property of the message is provided.

Message Integrity - *Message Integrity* is a property of the message and digital signature is the service or 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 cryptographic binding between a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved.

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



Security Tokens

Unsigned Security Tokens	Signed Security Tokens
→ Username	→ X.509 Certificates → Kerberos tickets

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

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.

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

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 the elements be "processed" this means that the element type be recognized well enough to return appropriate error if 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, an attachment, or any combination of them (or parts of them).

Message integrity is provided by leveraging XML Signature in conjunction with security tokens to ensure that messages are received without modifications. 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 also does not specify any signature appearing outside of `<wsse:Security>` element, if any.

3.3 Invalid or Missing Claims

The message recipient SHOULD reject a message with a signature determined to be invalid, missing or unacceptable claims as it is an 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 username security token containing a claimed security identity to establish a password derived signing key. The password is not provided in the security token. The message sender combines the password with the nonce and timestamp appearing in the security token to define an HMAC signing key that it then uses 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. The nonce and timestamp are used in the key calculation to introduce variability in the keys derived from a given password value.

```
<?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#">
  <S:Header>
     <wsse:Security
       xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
        <wsse:UsernameToken wsu:Id="MyID">
            <wsse:Username>Zoe</wsse:Username>
            <wsse:Nonce>FKJh...</wsse:Nonce>
            <wsu:Created>2001-10-13T09:00:00Z</wsu:Created>
        </wsse:UsernameToken>
        <ds:Signature>
           <ds:SignedInfo>
              <ds:CanonicalizationMethod
                  Algorithm=
                    "http://www.w3.org/2001/10/xml-exc-c14n#"/>
              <ds:SignatureMethod
                  Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
              <ds:Reference URI="#MsgBody">
                 <ds:DigestMethod
                    Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#sha1"/>
                 <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
              </ds:Reference>
           </ds:SignedInfo>
           <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
           <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                <wsse:Reference URI="#MyID"/>
               </wsse:SecurityTokenReference>
           </ds:KeyInfo>
        </ds:Signature>
     </wsse:Security>
  </S:Header>
  <S:Body wsu:Id="MsgBody">
    <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
        QQQ
    </tru:StockSymbol>
  </S:Body>
</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 (026)

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

Lines (010) to (025) 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 (011) to (018) describe what is being signed and the type of canonicalization being used. Line (012) specifies how to canonicalize (normalize) the data that is being signed. Lines (014) to (017) select the elements that are signed and how to digest them. Specifically, line (014) 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 (019) specifies the signature value of the canonicalized form of the data that is being signed as defined in the XML Signature specification.

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

Lines (028) to (030) contain the *body* (payload) of the SOAP message.

4 ID References

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

ID attributes from XML Signature

ID attributes from XML Encryption

wsu:Id global attribute described below

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

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.

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/2002/xx/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.

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 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}

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.

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 subject confirmation must be done, however, it does define how signatures can be used and associated with security tokens (by referencing them in the signature) as a form of Proof-of-Possession

6.2 User Name Tokens

6.2.1 Usernames and Passwords

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

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

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

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

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

That is, concatenate the nonce, creation timestamp, and the password (or shared secret or password equivalent) and include the digest of the combination. This helps obscure the password and offers a basis for preventing replay attacks. It is RECOMMENDED that timestamps and nonces 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.

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

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

The following illustrates the syntax of this element:

```
<wsse:UsernameToken wsu:Id="...">
    <wsse:Username>...</wsse:Username>
    <wsse:Password Type="...">...</wsse:Password>
    <wsse:Nonce EncodingType="...">...</wsse:Nonce>
    <wsu:Created>...</wsu:Created>
</wsse:UsernameToken>
```

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

/wsse:UsernameToken

This element is used for sending basic authentication information.

/wsse:UsernameToken/@wsu:Id

A string label for this security token.

/wsse:UsernameToken/Username

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

/wsse:UsernameToken/Username/@{any}

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

/wsse:UsernameToken/Password

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

/wsse:UsernameToken/Password/@Type

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

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

/wsse:UsernameToken/Password/@{any}

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

/wsse:UsernameToken//wsse:Nonce

This optional element specifies a cryptographically random nonce.

/wsse:UsernameToken//wsse:Nonce/@EncodingType

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

/wsse:UsernameToken//wsu:Created

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

/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 header.

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

The following illustrates the use of this element (note that in this example the password is sent in clear text and the message should therefore be sent over a confidential channel:

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Zoe</wsse:Username>
                <wsse:Password>ILoveDogs</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

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

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext">
    <S:Header>
            ...
        <wsse:Security>
          <wsse:UsernameToken
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
            <wsse:Username>NNK</wsse:Username>
            <wsse:Password Type="wsse:PasswordDigest">
                FEdR...</wsse:Password>
            <wsse:Nonce>FKJh...</wsse:Nonce>
            <wsu:Created>2001-10-13T09:00:00Z </wsu:Created>
          </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 using XML namespaces. Subsequent specifications MUST define the ValueType value for the tokens that they define.

/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 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	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 support 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/2002/xx/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. Subsequent 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 binding documents for the appropriate reference mechanism. However, specific extensions MAY be made to the `wsse:SecurityTokenReference>` element.

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.

The usage of SecurityTokenRefeference used outside of the `<Security>` header block is unspecified.

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.

/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.

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. 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. 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 unique.

The following list provides a list of the specific reference mechanisms defined in WS-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 bereferenced 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.

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.

/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.

/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/2002/xx/secext">
   <wsse:Reference
             URI="http://www.fabrikam123.com/tokens/Zoe#X509token"/>
```

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

/wsse:SecurityTokenReference/KeyIdentifier/@EncodingType

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

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 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 binding 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>
```

7.5 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, defined for e-mail addresses, SHOULD conform to RFC 822:

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

7.6 Token Reference Lookup Processing Order

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

1. Resolve any `<wsse:Reference>` elements (specified within `<wsse:SecurityTokenReference>`).
2. Resolve any `<wsse:KeyIdentifier>` elements (specified within `<wsse:SecurityTokenReference>`).
3. Resolve any `<ds:KeyName>` elements.
4. Resolve any other `<ds:KeyInfo>` elements.

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

8 Signatures

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

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

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#
Transformations	XML Decryption Transformation	http://www.w3.org/2001/04/decrypt#

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 use the Decryption Transformation for XML Signature.

8.2 Signing Messages

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

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

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

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

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

8.3 Signature Validation

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

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.

8.4 Example

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

```
<?xml version="1.0" encoding="utf-8"?>
<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/2002/xx/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#"/>
               <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>
```

9 Encryption

This specification allows encryption of any combination of body blocks, header blocks, any of these sub-structures, and attachments 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 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 prepend the sub-element into the `<wsse:Security>` header block for the targeted recipient that is expected to decrypt these encrypted portions. The combined process of encrypting portion(s) of a message and adding one of these a sub-elements referring to the encrypted portion(s) is called an encryption step hereafter. The sub-element should contain enough information for the recipient to identify which portions of the message are to be decrypted by the recipient.

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

9.1 xenc:ReferenceList

When encrypting elements or element contents within a SOAP envelope, 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 an `<xenc:ReferenceList>` element.

Although in XML Encryption, `<xenc:ReferenceList>` is 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/2002/xx/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>
            <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/2002/xx/secext"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
        <wsse:Security>
            <xenc:EncryptedKey>
               <xenc:EncryptionMethod Algorithm="..."/>
               <ds:KeyInfo>
                  <wsse:SecurityTokenReference>
               <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
                     ValueType="wsse:X509v3">MIGfMa0GCSq...
               </wsse:KeyIdentifier>
                  </wsse:SecurityTokenReference>
               </ds:KeyInfo>
               <xenc:CipherData>
                   <xenc:CipherValue>...</xenc:CipherValue>
               </xenc:CipherData>
               <xenc:ReferenceList>
                  <xenc:DataReference URI="#bodyID"/>
               </xenc:ReferenceList>
            </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>
```

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 xenc:EncryptedData

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

The contents of the attachment MUST be replaced by the encrypted octet string.

The replaced MIME part MUST have the media type `application/octet-stream`.

The original media type of the attachment MUST be declared in the `MimeType` attribute of the `<xenc:EncryptedData>` element.

The encrypted MIME part MUST be referenced by an `<xenc:CipherReference>` element with a URI that points to the MIME part with `cid:` as the scheme component of the URI.

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

```
<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/2002/xx/secext"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
        <wsse:Security>
            <xenc:EncryptedData MimeType="image/png">
            <ds:KeyInfo>
                  <wsse:SecurityTokenReference>
               <xenc:EncryptionMethod Algorithm="..."/>
               <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
                     ValueType="wsse:X509v3">MIGfMa0GCSq...
               </wsse:KeyIdentifier>
                  </wsse:SecurityTokenReference>
               </ds:KeyInfo>
               <xenc:CipherData>
                   <xenc:CipherReference URI="cid:image"/>
               </xenc:CipherData>
            </xenc:EncryptedData>
        </wsse:Security>
    </S:Header>
    <S:Body> </S:Body>
</S:Envelope>
```

9.4 Processing Rules

Encrypted parts or attachments to the SOAP message 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. of the contents of `<S:Body>`) 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. This specification allows placing the encrypted octet stream in an attachment. For example, if an `<xenc:EncryptedData>` element in an `<S:Body>` element has `<xenc:CipherReference>` that refers to an attachment, then the decrypted octet stream SHALL replace the `<xenc:EncryptedData>`. However, if the `<enc:EncryptedData>`

element is located in the <Security> header block and it refers to an attachment, then the decrypted octet stream MUST replace the encrypted octet stream in the attachment.

9.4.1 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, and attachments.

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. For an attachment, the contents MUST be replaced by encrypted cipher data as described in section 9.3 Signature Validation.

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>.

9.4.2 Decryption

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

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

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

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.

9.5 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 Message Timestamps

It is often important for the recipient to be able to determine the *freshness* of a message. In some cases, a message 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 either that the recipient is using a mechanism to synchronize time (e.g. NTP) or, more likely for federated applications, that they are making assessments about time based on three factors: creation time of the message, transmission checkpoints, and transmission delays and their local time.

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

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

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. 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. Implementations MUST NOT generate time instants that specify leap seconds.

10.1 Model

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

10.2 Timestamp Elements

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

10.2.1 Creation

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

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

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

/wsu:Created

This element's value is a creation timestamp. Its type is specified by the ValueType attribute.

/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:Created/@wsu:Id

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

10.2.2 Expiration

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

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

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

/wsu:Expires

This element's value represents an expiration time. Its type is specified by the ValueType attribute

/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:Expires/@wsu:Id

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

One suggested formula for estimating clock skew is

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

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

10.3 Timestamp Header

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

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.

Multiple `<wsu:Timestamp>` headers can be specified if they are targeted at different SOAP roles. The ordering within the header is as illustrated below.

The ordering of elements in this header is fixed and MUST be preserved by intermediaries.

To preserve overall integrity of each `<wsu:Timestamp>` header, it is strongly RECOMMENDED that each SOAP role create or update the appropriate `<wsu:Timestamp>` header destined to itself.

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

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

This represents the creation time of the message. This element is optional, but can only be specified once in a `Timestamp` header. 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/Expires

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

/wsu:Timestamp/{any}

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

/wsu:Timestamp/@wsu:Id

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

/wsu:Timestamp/@{any}

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

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:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
  <S:Header>
    <wsu:Timestamp>
       <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
       <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
    </wsu:Timestamp>
    ...
  </S:Header>
  <S:Body>
```

```
      ...
    </S:Body>
</S:Envelope>
```

10.4 TimestampTrace Header

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

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.

Multiple `<wsu:TimestampTrace>` headers can be specified if they reference a different SOAP role.

The `<wsu:Received>` element specifies a receipt timestamp with an optional processing delay. The exact meaning and semantics are dependent on the context in which the element is used.

It is also strongly RECOMMENDED that each SOAP role sign its elements by referencing their ID, NOT by signing the `TimestampTrace` header as the header is mutable.

The syntax for this element is as follows:

```
<wsu:TimestampTrace>
   <wsu:Received Role="..." Delay="..." ValueType="..."
                 wsu:Id="...">...</wsu:Received>
</wsu:TimestampTrace>
```

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

/wsu:Received

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

/wsu:Received/@Role

A required attribute, `Role`, indicates which SOAP role is indicating receipt. Roles MUST include this attribute, with a value matching the role value as specified as a SOAP intermediary.

/wsu:Received/@Delay

The value of this optional attribute is the delay associated with the SOAP role expressed in milliseconds. The delay represents processing time by the Role after it received the message, but before it forwarded to the next recipient.

/wsu:Received/@ValueType

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

/wsu:Received/@wsu:Id

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

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

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

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

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility">
  <S:Header>
    <wsu:Timestamp>
       <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
       <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
    </wsu:Timestamp>
    <wsu:TimespampTrace>
       <wsu:Received Role="http://x.com/" Delay="60000">
                2001-09-13T08:44:00Z</wsu:Received>
    </wsu:TimestampTrace>
    ...
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>

```

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/2002/xx/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/xx/utility"
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
(003)   <S:Header>
(004)       <wsu:Timestamp>
(005)           <wsu:Created wsu:Id="T0">
(006)               2001-09-13T08:42:00Z
(007)           </wsu:Created>
(008)       </wsu:Timestamp>
(009)      <wsse:Security>
(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)             <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
(016)                ValueType="wsse:X509v3">MIGfMa0GCSq...
(017)             </wsse:KeyIdentifier>
(018)             <xenc:CipherData>
(019)                <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(020)                </xenc:CipherValue>
(021)             </xenc:CipherData>
(022)             <xenc:ReferenceList>
(023)                 <xenc:DataReference URI="#enc1"/>
(024)             </xenc:ReferenceList>
(025)         </xenc:EncryptedKey>
(026)         <ds:Signature>
(027)            <ds:SignedInfo>
(028)               <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(029)               <ds:SignatureMethod
         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(039)               <ds:Reference URI="#T0">
(031)                  <ds:Transforms>
(032)                     <ds:Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(033)                  </ds:Transforms>
(034)                  <ds:DigestMethod
             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(035)                  <ds:DigestValue>LyLsF094hPi4wPU...
(036)                   </ds:DigestValue>
(037)               </ds:Reference>
(038)               <ds:Reference URI="#body">
(039)                  <ds:Transforms>
(040)                     <ds:Transform
```

```
                  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(041)                  </ds:Transforms>
(042)                  <ds:DigestMethod
                   Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(043)                  <ds:DigestValue>LyLsF094hPi4wPU...
(044)                   </ds:DigestValue>
(045)               </ds:Reference>
(046)            </ds:SignedInfo>
(047)            <ds:SignatureValue>
(048)                     Hp1ZkmFZ/2kQLXDJbchm5gK...
(049)            </ds:SignatureValue>
(050)            <ds:KeyInfo>
(051)                <wsse:SecurityTokenReference>
(052)                    <wsse:Reference URI="#X509Token"/>
(053)                </wsse:SecurityTokenReference>
(054)            </ds:KeyInfo>
(055)         </ds:Signature>
(056)      </wsse:Security>
(057)   </S:Header>
(058)   <S:Body wsu:Id="body">
(059)      <xenc:EncryptedData
                  Type="http://www.w3.org/2001/04/xmlenc#Element"
                  wsu:Id="enc1">
(060)         <xenc:EncryptionMethod
              Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
(061)         <xenc:CipherData>
(062)            <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(063)            </xenc:CipherValue>
(064)         </xenc:CipherData>
(065)      </xenc:EncryptedData>
(066)   </S:Body>
(067) </S:Envelope>
```

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

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

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

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

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)-(025) 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)-(017) specify the name of the key that was used to encrypt the symmetric key. Lines (018)-(021) specify the actual encrypted form of the symmetric key. Lines (022)-(024) 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").

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

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

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

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

Lines (059)-(065) represent the encrypt ed metadata and form of the body using XML Encryption. Line (059) indicates that the "element value" is being replaced and identifies this encryption. Line

(060) specifies the encryption algorithm – Triple-DES in this case. Lines (062)-(063) 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 (023).

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

These can be grouped into two *classes* of errors: unsupported and failure. For the case of unsupported errors, the recipient MAY provide a response that informs the sender of supported formats, etc. For failure errors, the recipient MAY choose not to respond, as this may be a form of Denial of Service (DOS) 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 SOAP's 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

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 aginst replay attacts.

When digital signatures are used for verifying the identity of the sending party, the sender must prove the possession of the private 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.

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

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 aginst replay attacts.

In order to *trust* Ids and timestamps, 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 nonces 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 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 Section 6.1.1. Even so, the password must be strong enough so that simple password guessing attacks will not reveal the secret from a captured message.

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 RECOMMEND that the nonce value be unique per message (never been used as a nonce before by the sender and recipient) and use the `<wsse:Nonce>` element 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.

14 Privacy Considerations

TBD

15 Acknowledgements

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

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

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

[XML-Encrypt] W3C Working Draft, "XML Encryption Syntax and Processing," 04 March 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 02, "Web Services Security SAML Token Binding, 23 September 2002

[WSS-XrML] OASIS Working Draft 01, "Web Services Security XrML Token Binding, 20 September 2002

[WSS-X509] OASIS Working Draft 01, "Web Services Security X509 Binding, 18 September 2002

[WSS-Kerberos] OASIS Working Draft 01, "Web Services Security Kerberos Binding, 18 September 2002

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

Appendix A: Utility Elements and Attributes

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

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

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, expiration, and receipt. While defined in the context of messages, these elements can be re-used wherever these sorts of time statements need to be made.

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

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

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 dateTime type to include the common attributes.

Appendix B: SecurityTokenReference Model

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

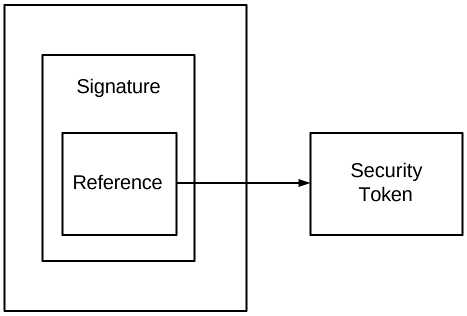
There are several motivations for introducing the `<wsse:SecurityTokenReference>` element:

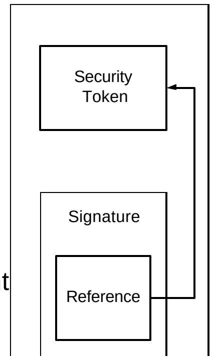
The XML Signature reference mechanisms are focused on "key" references rather than general token references.
The XML Signature reference mechanisms utilize a fairly closed schema which limits the extensibility that can be applied.
There are additional types of general reference mechanisms that are needed, 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:

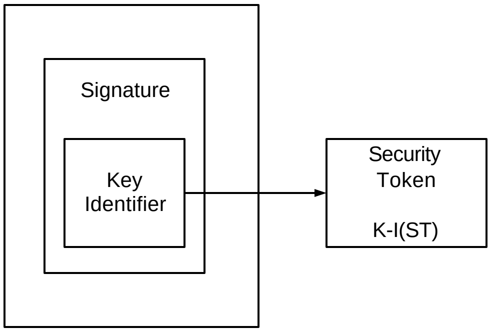
Local Reference – A security token, that is included in the message in the `<wsse:Security>` header, is associated with an XML Signature. The figure below illustrates this:

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:

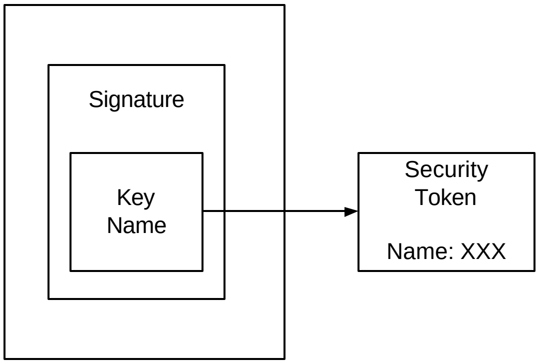




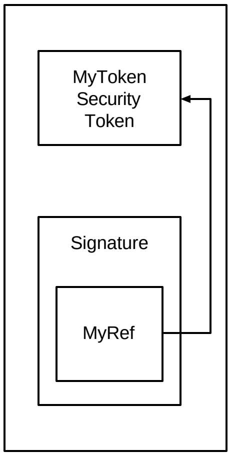
Key Identifier – A security token, which is associated with an XML Signature and identified using a known value that is the result of a well-known function of the security token (defined by the token format or profile). The figure below illustrates this where the token is located externally:



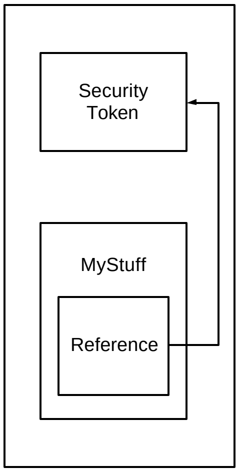
Key Name – A security token is associated with an XML Signature and identified using a known value that represents a "name" assertion within the security token (defined by the token format or profile). The figure below illustrates this where the token is located externally:



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:



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 WS-Security or its profiles are preferred over the mechanisms in XML Signature.

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

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

Signature doesn't imply formatting semantics, WS-Security RECOMMENDS that X.509 names be specified.

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

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

1721
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

1722

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 implementors 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.