Web Services Security
Core Specification

Working Draft 05, 02 December 2002

Document identifier:
WSS-Core-04

Location:
TBD

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

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 Nagarathnam, 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, message confidentiality, 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.shtm).
# Table of Contents

1  Introduction .................................................................................................5

1.1 Goals and Requirements .........................................................................5

1.1.1 Requirements ......................................................................................5

1.1.2 Non-Goals ..........................................................................................5

2  Notations and Terminology ........................................................................7

2.1 Notational Conventions .........................................................................7

2.2 Namespaces ............................................................................................7

2.3 Terminology ............................................................................................7

3  Message Protection Mechanisms .............................................................9

3.1 Message Security Model .........................................................................9

3.2 Message Protection .................................................................................9

3.3 Invalid or Missing Claims ......................................................................10

3.4 Example ..................................................................................................10

4  ID References ............................................................................................12

4.1 Id Attribute ............................................................................................12

4.2 Id Schema ...............................................................................................12

5  Security Header ........................................................................................14

6  Security Tokens ........................................................................................16

6.1 User Name Tokens ................................................................................16

6.2 Binary Security Tokens ..........................................................................18

6.2.1 Attaching Security Tokens ...............................................................18

6.2.2 Processing Rules ...............................................................................18

6.2.3 Encoding Binary Security Tokens ....................................................18

6.3 XML Tokens ..........................................................................................19

6.3.1 Attaching Security Tokens ...............................................................20

6.3.2 Identifying and Referencing Security Tokens ................................20

6.3.3 Subject Confirmation ........................................................................20

6.3.4 Processing Rules ...............................................................................20

7  Token References .......................................................................................21

7.1 SecurityTokenReference Element .......................................................21

7.2 Direct References ..................................................................................21

7.3 Key Identifiers .......................................................................................22

7.4 ds:KeyInfo .............................................................................................23

7.5 Key Names .............................................................................................23

7.6 Token Reference Lookup Processing Order .......................................23

8  Signatures ..................................................................................................25

8.1 Algorithms ...............................................................................................25

8.2 Signing Messages.....................................................................................26

8.3 Signature Validation ...............................................................................26

8.4 Example ...................................................................................................26
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 hierarchy associated with the keys used for signing and encryption).

1.1 Goals and Requirements

The goal of this specification is to enable applications to construct 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 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") represent some application-dependent or context-dependent URI as defined in RFC2396.

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):


The following namespaces are used in this document:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td><a href="http://www.w3.org/2001/12/soap-envelope">http://www.w3.org/2001/12/soap-envelope</a></td>
</tr>
<tr>
<td>ds</td>
<td><a href="http://www.w3.org/2000/09/xmldsig#">http://www.w3.org/2000/09/xmldsig#</a></td>
</tr>
<tr>
<td>xenc</td>
<td><a href="http://www.w3.org/2001/04/xmlenc#">http://www.w3.org/2001/04/xmlenc#</a></td>
</tr>
</tbody>
</table>

2.3 Terminology

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

- **Claim** – A claim is a declaration made by a client, (e.g. name, identity, key, group, privilege, capability, etc).
- **Security Token** – A security token represents a collection of claims.
- **Signed Security Token** – A signed security token is a security token that is asserted and cryptographically signed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket).
### Proof-of-Possession

Proof-of-possession information, authentication data that is provided with a message to prove that the message was sent and/or created by a claimed identity based on knowledge of information that should only be known to the claimed identity.

### Integrity

Integrity is the property that data has not been modified.

### 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.

### Confidentiality

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

### 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.

### Digest

A digest is a cryptographic checksum of an octet stream.

### Signature

A signature is a cryptographic binding of 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.

### 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.

### 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.

### 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, i.e., companies, divisions, 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.

---

### Security Tokens

<table>
<thead>
<tr>
<th>Unsigned Security Tokens</th>
<th>Signed Security Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ Username</td>
<td>→ X.509 Certificates</td>
</tr>
<tr>
<td></td>
<td>→ Kerberos tickets</td>
</tr>
</tbody>
</table>
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 the security token and thus authenticate the claims in the security token. An X.509 certificate, claiming the binding between one's identity and public key, is an example of a signed security token, and thus endorsed by the certificate authority, security token. In the absence of endorsement by a third party, the recipient of a security token may chose 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.

A claim can be either signed or unsigned by a trusted authority. A set of signed claims is usually represented as a signed security token that is digitally signed or encrypted by the authority. An X.509 certificate, claiming the binding between one's identity and public key, is an example of a signed security token. An signed claim can also be represented as a reference to an authority so that the recipient can "pull" the claim from the referenced authority.

An unsigned claim can be trusted if there is a trust relationship between the sender and the recipient. For example, the unsigned claim that the sender is Bob is sufficient for a certain recipient to believe that the sender is in fact Bob, if the sender and the recipient use a trusted connection and there is an out-of-band trust relationship between them.

One special type of unsigned claim is Proof-of-Possession. Such a claim proves that the sender has a particular piece of knowledge that is verifiable by appropriate SOAP roles. For example, a username/password is a security token with this type of claim. A Proof-of-Possession claim is sometimes combined with other security tokens to prove the claims of the sender. Note that a digital signature used for message integrity can also be used as a Proof-of-Possession claim, although this specification does not consider such a digital signature as a type of security token.

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.

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 transmitted 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.


### 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: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: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:Envelope>
```

Deleted: The following example illustrates a message with a username security token. In this example the password is not provided in plaintext. Instead, it is used as a "shared secret" which can be used, for example, as part of an HMAC signature to authenticate messages. The exact algorithm is out-of-scope of this specification, however, in the example below, the information inside the <UsernameToken> element is concatenated with the key so as to include random elements (nonce and timestamp). In some cases, the nonce may be provided as a challenge using some out-of-band mechanism.
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 (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 (021) 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 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:

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

The following describes the attribute illustrated above:

```xml
.../@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:
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 XML Schema or DTDs are strongly encouraged 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." Specifically, implementations MAY support the value of the \texttt{wsu:Id} as the valid identifier for use as an XPointer shorthand pointer.
5 Security Header

The `<wsse:Security>` header block provides a mechanism for attaching security-related information targeted at a specific recipient (SOAP role). This MAY be either the ultimate recipient of the message or an intermediary. Consequently, this header block 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 the same 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 can omit the `S:role` attribute and no two `<wsse:Security>` header blocks can 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` can 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 policy is needed.

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="..."/>
    ...
  </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.
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 profile.

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

This chapter discusses different types of security tokens and how they are attached to messages.

6.1 User Name Tokens

6.1.1 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 only 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:

Password_digest = 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>
  <wsse:Username>Zoe</wsse:Username>
  <wsse:Password>ILoveDogs</wsse:Password>
</wsse:UsernameToken>
```

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

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsse:PasswordText</td>
<td>The actual password for the username or derived password or S/KEY.</td>
</tr>
<tr>
<td>wsse:PasswordDigest</td>
<td>The digest of the password for the username using the algorithm described above.</td>
</tr>
</tbody>
</table>

This optional element specifies a cryptographically random nonce.

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.

This optional element which specifies a timestamp.

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

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):

```
<宋:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  <S:Header>
    ...
  </S:Header>
  <S:Body>
    <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>Zoe</wsse:Username>
        <wsse:Password>ILoveDogs</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
    ...
  </S:Body>
</S:Envelope>
```
The following example illustrates a hashed password using both a nonce and a timestamp with the password hashed:

```
<Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  <Header>
    ...
  </Header>
  <wsse:Security>
    <wsse:UsernameToken
      <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>
  ...
</Envelope>
```

6.2 Binary Security Tokens

6.2.1 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.

6.2.2 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 including XML-based tokens. Note that this does NOT mean that binary security tokens MUST be signed or encrypted – only that if signature or encryption is used in conjunction with binary security tokens, they MUST be used in a way that conforms to the processing rules defined by this specification.

6.2.3 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 describe rules and processes for 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:
This element is used to include a binary-encoded security token.

An optional string label for this security token.

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.

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

<table>
<thead>
<tr>
<th>QName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>wsse:Base64Binary</code></td>
<td>XML Schema base 64 encoding</td>
</tr>
</tbody>
</table>

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 are 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.

```xml
<wsse:BinarySecurityToken
    wsu:Id="myToken"
    ValueType="x:MyType" xmlns:x="http://www.fabrikam123.com/x"
    EncodingType="wsse:Base64Binary">
    MIIEZzCCA9GwIBAgIEjE0...
</wsse:BinarySecurityToken>
```

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.
6.3.1 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.

6.3.2 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.

6.3.3 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.

6.3.4 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 including XML-based tokens. Note that this does NOT mean that XML-based tokens MUST be signed or encrypted – only that if signature or encryption is used in conjunction with XML-based tokens, they MUST be used in a way that conforms to the processing rules defined by this specification.
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 following illustrates the syntax of this element:

```xml
<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/{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.

The following illustrates the use of this element:

```xml
<wsse:SecurityTokenReference
  <wsse:Reference
    URI="http://www.fabrikam123.com/tokens/Zoe#X509token"/>
</wsse:SecurityTokenReference>
```

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.

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:

```xml
<wsse:SecurityTokenReference wsu:Id="...">
  ...
</wsse:SecurityTokenReference>
```
The following describes the elements defined above:

- `<wsse:SecurityTokenReference/Reference>`
  
  This element is used to identify a URI location for locating a security token.

- `<wsse:SecurityTokenReference/Reference/@URI>`
  
  This optional attribute specifies a 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, specification for individual token types MAY define specific processing rules and semantics around the value of the URI and how it is interpreted. If this attribute is not present, the URI is 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:

```xml
<wsse:SecurityTokenReference
  <wsse:Reference
    URI="http://www.fabrikam123.com/tokens/Zoe#X509token"/>
</wsse:SecurityTokenReference>
```

### 7.3 Key Identifiers

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

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:

```xml
<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:

<table>
<thead>
<tr>
<th>QName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsse:Base64Binary</td>
<td>XML Schema base 64 encoding (default)</td>
</tr>
</tbody>
</table>

/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:

```xml
<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 **RECOMMEND** 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 are the following convention for e-mail addresses, which **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.
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.

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

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 desired 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 subsystem 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 did what.

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:

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

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 is 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:

1. The application MUST be capable of processing the required elements defined in the XML Signature specification.
2. To add a signature to a <wsse:Security> header block, a <ds:Signature> element conforming to the XML Signature specification SHOULD be prepended to the existing content of the <wsse:Security> header block. That is, the new information would be before (prepended to) the old. 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 fails if
1. the syntax of the content of the entry does not conform to this specification, or
2. the validation of the signature contained in the entry fails according to the core validation of the XML Signature specification, or
3. the application applying its own validation policy rejects the message for some reason (e.g., the signature is created by an untrusted key – verifying the previous two steps only performs cryptographic verification of the signature).

If the verification of the signature entry 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, we sign only the message body.

```xml
<?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: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/vXcMNjPOV...
        </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 key carried in the message in an encrypted form.

In order to allow this flexibility, this specification leverages the XML Encryption standard. Specifically, described 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 add 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 sub-elements referring to the encrypted portion(s) is called an encryption step hereafter. The sub-element should have 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: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:EncryptedData>
</S:Body>
```
9.2 xenc:EncryptedKey

When the encryption step involves encrypting elements or element contents within a SOAP envelope with a key, which is in turn to be encrypted by the recipient’s key and embedded in the message, $\texttt{xenc:EncryptedKey}$ MAY be used for carrying such an encrypted key. This sub-element SHOULD have a manifest, that is, an $\texttt{xenc:ReferenceList}$ element, in order for the recipient to know the portions to be decrypted with this key (if any exist). An element or element content to be encrypted by this encryption step MUST be replaced by a corresponding $\texttt{xenc:EncryptedData}$ element according to XML Encryption. All the $\texttt{xenc:EncryptedData}$ elements created by this encryption step SHOULD be listed in the $\texttt{xenc:ReferenceList}$ 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:

```xml
<S:Envelope
  xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  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...</n
Comment: A naked $\texttt{wsse:KeyIdentifier}$ would be illegal.
```

While XML Encryption specifies that $\texttt{xenc:EncryptedKey}$ elements MAY be specified in $\texttt{xenc:EncryptedData}$ elements, this specification strongly RECOMMENDS that $\texttt{xenc:EncryptedKey}$ elements be placed in the $\texttt{wsse:Security}$ 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 can 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).

1. The contents of the attachment MUST be replaced by the encrypted octet string.
2. The replaced MIME part MUST have the media type `application/octet-stream`.
3. The original media type of the attachment MUST be declared in the `MimeType` attribute of the `<xenc:EncryptedData>` element.
4. 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: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>
    </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>` appearing inside the `<S:Body>` element has `<xenc:CipherReference>` that refers to an attachment, then the decrypted octet...
stream SHALL replace the <xenc:EncryptedData>. However, if the <xenc: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).

1. Create a new SOAP envelope.

2. 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.

3. Locate data items to be encrypted, i.e., XML elements, element contents within the target
SOAP envelope, and attachments.

4. 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 8.3 Signature Validation.

5. 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.

6. 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 with encryption header entries, for each encryption header entry
the following general steps should be processed (non-normative):

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

2. 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.

3. 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 takes place in another <wsse:Security>
header, the order may not be explicitly understood.
If the sender wishes to sign a message that is subsequently encrypted by an intermediary along the transmission path, the sender MAY use the Decryption Transform for XML Signature to explicitly specify the order of decryption.
10 Message Timestamps

When requestors and services are exchanging messages, it is often important to be able to understand 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.

To assist a recipient in making an assessment of staleness, a requestor may wish to indicate a suggested expiration time, beyond which the requestor recommends ignoring 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.

10.1 Model

This specification provides several tools for recipients to use to assess the expiration time presented by the requestor. The first is the creation time. Recipients can use this value to assess possible clock synchronization issues. 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 trust.

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 intermediary markers are needed.

10.2.1 Expiration

The <wsu:Expires> element specifies the expiration timestamp. 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:

```xml
<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. The time specified SHOULD be a UTC format as specified by the ValueType attribute (default is XML Schema type dateTime).

/\wsu:Expires/@ValueType

This optional attribute specifies the type of the time data. This is specified as the XML Schema type. If this attribute isn't specified, 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, will need to make a 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 synchronization.

One suggested formula for estimating synchronization 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.

10.2.2 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:

\(<\wsu:Created\ Value\ Type=...\>\ wsu:Id=...\>...\</\wsu:Created>\)

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

/\wsu:Created

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

/\wsu:Created/@ValueType

This optional attribute specifies the type of the time data. This is specified as the XML Schema type. If this attribute isn't specified, 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.
10.3 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, it 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:

```xml
<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.

```xml
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  <S:Header>
    ...
  </S:Header>
</S:Envelope>
```
A `<wsu:TimestampTrace>` header provides a mechanism for expressing the delays introduced throughout the message path. Specifically, it 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.
- `/wsu:Received/@wsu:Id`
  - 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.

```xml
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  <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:TimestampTrace>
      <wsu:Received Role="http://x.com/" Delay="60000"/>
      <wsu:Created>2001-09-13T08:44:00Z</wsu:Created>
      <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
    </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.

```
<?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:xenc="http://www.w3.org/2001/04/xmlenc#">
  <S:Header>
    <wsu:Timestamp>
      <wsu:Created wsu:Id="T0">
        2001-09-13T08:42:00Z
      </wsu:Created>
    </wsu:Timestamp>
    <wsse:Security>
      <wsse:BinarySecurityToken
            ValueType="wsse:X509v3"
            wsu:Id="X509Token"
            EncodingType="wsse:Base64Binary">
        MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <wsse:KeyIdentifier EncodingType="wsse:Base64Binary" Value="wsse:X509v3">MIGfMa0GCSq...
        <xenc:KeyIdentifier/>
      </xenc:EncryptedKey>
      <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="#T0">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            </ds:Transforms>
          </ds:Reference>
          <ds:Reference URI="#body">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            </ds:Transforms>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>LyLsFi094hP14wPU... </ds:SignatureValue>
      </ds:Signature>
    </wsse:Security>
  </S:Header>
</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) provides a URI link to the Lines (010)-(012).

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

Lines (059)-(065) represent the encrypted metadata and form of the body using XML Encryption.

Line (059) indicates that the "element value" is being replaced and identifies this encryption.
(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

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:

<table>
<thead>
<tr>
<th>Error that occurred</th>
<th>faultcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>An unsupported token was provided</td>
<td>wsse:UnsupportedSecurityToken</td>
</tr>
<tr>
<td>An unsupported signature or encryption algorithm was used</td>
<td>wsse:UnsupportedAlgorithm</td>
</tr>
</tbody>
</table>

The "failure" class of errors are:

<table>
<thead>
<tr>
<th>Error that occurred</th>
<th>faultcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>An error was discovered processing the (<a href="">wsse:Security</a>) header.</td>
<td>wsse:InvalidSecurity</td>
</tr>
<tr>
<td>An invalid security token was provided</td>
<td>wsse:InvalidSecurityToken</td>
</tr>
<tr>
<td>The security token could not be authenticated or authorized</td>
<td>wsse:FailedAuthentication</td>
</tr>
<tr>
<td>The signature or decryption was invalid</td>
<td>wsse:FailedCheck</td>
</tr>
<tr>
<td>Referenced security token could not be retrieved</td>
<td>wsse:SecurityTokenUnavailable</td>
</tr>
</tbody>
</table>
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).

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.

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

In order to trust 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


[WS-Security Addendum], IBM, Microsoft, VeriSign, August 2002.

[WS-Security XML Tokens], IBM, Microsoft, VeriSign, August 2002.


Appendix A: Revision History

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>20-Sep-02</td>
<td>Initial draft based on input documents and editorial review</td>
</tr>
<tr>
<td>02</td>
<td>24-Oct-02</td>
<td>Update with initial comments (technical and grammatical)</td>
</tr>
<tr>
<td>03</td>
<td>03-Nov-02</td>
<td>Feedback updates</td>
</tr>
<tr>
<td>04</td>
<td>17-Nov-02</td>
<td>Feedback updates</td>
</tr>
</tbody>
</table>
Appendix B: 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.

Copyright © OASIS Open 2002. All Rights Reserved.

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

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

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