Web Services Security (WS-Security)

Version 1.0 05 April 2002

This version:
http://www.ibm.com/developerworks/library/ws-secure/

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

Editor:
Chris Kaler, Microsoft

Abstract

WS-Security describes enhancements to 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.

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

Additionally, WS-Security describes how to encode binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the credentials that are included with a message.

Composable Architecture

By using the SOAP extensibility model, SOAP-based specifications are designed to be composed with each other to provide a rich messaging environment. By itself, WS-Security does not ensure security nor does it provide a complete security solution. WS-Security is a building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models and encryption technologies. Implementing WS-Security does not mean that an application cannot be attacked or that the security cannot be compromised.

Status of this Document

WS-Security and related specifications are provided as-is and for review and evaluation only. IBM and Microsoft and VeriSign hope to solicit your contributions and suggestions in the near future. IBM and Microsoft and Versign make no warrantees or representations regarding the specifications in any manner whatsoever.

Table of Contents

1. Introduction

This specification proposes a standard set of SOAP extensions that can be used when building secure Web services to implement integrity and confidentiality. We refer to this set of extensions as the â€œWeb Services Security Language" or â€œWS-Security".

WS-Security is flexible and is designed to be used as the basis for the construction of a wide variety of security models including PKI, Kerberos, and SSL. Specifically WS-Security provides support for multiple security tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies.

This specification provides three main mechanisms: security token propagation, message integrity, and message confidentiality. These mechanisms by themselves do not provide a complete security solution. Instead, WS-Security 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 encryption technologies.

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

This document supercedes existing web services security specifications from IBM and Microsoft including SOAP-SEC; Microsoft's WS-Security and WS-License; and IBM's security token and encryption documents.

Note that Section 1 is non-normative.

1.1. Goals and Requirements

The goal of WS-Security 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 WS-Security are not vulnerable to a wide range of attacks.

To summarize, 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 tokens for authentication or authorization

- Multiple trust domains
- 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 that require multiple exchanges.
- Key exchange and derived keys
- How trust is established or determined.

1.2. Example

The following example illustrates a message with a username security token:

```
<?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>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
        <m:action>http://fabrikam123.com/getQuote</m:action>
        <m:to>http://fabrikam123.com/stocks</m:to>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
      <wsse:Security
       xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
        wsse:UsernameToken Id="MyID">
            <wsse:Username>Zoe</wsse:Username>
        </wsse:UsernameToken>
        <ds:Signature>
           <ds:SignedInfo>
              <ds:CanonicalizationMethod
             Algorithm=
                    "http://www.w3.org/2001/10/xml-exc-c14n#"/>
              <ds:SignatureMethod
                  Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
              <ds:Reference URI="#MsgBody">
                 <ds:DigestMethod
                    Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#sha1"/>
                 <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
              </ds:Reference>
           </ds:SignedInfo>
           <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
           <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                <wsse:Reference URI="#MyID"/>
               </wsse:SecurityTokenReference>
           </ds:KeyInfo>
        </ds:Signature>
     </wsse:Security>
  </S:Header>
  <S:Body Id="MsgBody">
    <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
```

```
                QQQ
            </tru:StockSymbol>
(033)   </S:Body>
(034) </S:Envelope>
```

The first two lines start the SOAP envelope. Line (003) begins the headers that are associated with this SOAP message. Lines (004) to (008) specify how to route this message (as defined in WS-Routing).

Line (009) starts the <Security> header that we define in this specification. This header contains security information for an intended receiver. This element continues until line (029)

Lines (010) to (012) 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 we assume the service knows the password - in other words, it is a shared secret.

Lines (013) to (028) specify a digital signature. This signature ensures the integrity of the signed elements (that they aren't modified). The signature uses the XML Signature specification. In this example, the signature is based on a key generated from the users' password; typically stronger signing mechanisms would be used (see the Extended Example below).

Lines (014) to (021) describe the digital signature. Line (015) specifies how to canonicalize (normalize) the data that is being signed.

Lines (017) to (020) select the elements that are signed. Specifically, line (017) indicates that the <S:Body> element is signed. In this example only the message body is signed; typically additional elements of the message, such as parts of the routing header, should be included in the signature (see the Extended Example below).

Line (022) specifies the signature value of the canonicalized form of the data that is being signed as defined in the XML Signature specification.

Lines (023) to (027) provide a *hint* as to where to find the security token associated with this signature. Specifically, lines (024) to (025) indicate that the security token can be found at (pulled from) the specified URL.

Lines (031) to (033) contain the *body* (payload) of the SOAP message.

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.

WS-Security is designed to work with the general SOAP message structure and message processing model, and WS-Security 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 URI that MUST be used by implementations of this specification is:

```
http://schemas.xmlsoap.org/ws/2002/04/secext
```

The following namespaces are used in this document:

Prefix	Namespace
S	http://www.w3.org/2001/12/soap-envelope

ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#
m	http://schemas.xmlsoap.org/rp
wsse	http://schemas.xmlsoap.org/ws/2002/04/secext

2.3. Terminology

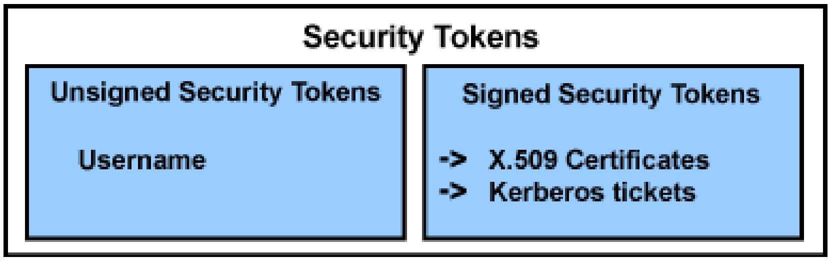
We provide basic definitions for the security terminology used in this specification.

Claim - A *claim* is a statement that a client makes (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 endorsed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket).

Figure 1



Proof-of-Possession - The *proof-of-possession* information is data that is used in a proof process to demonstrate the sender's knowledge of information that SHOULD only be known to the claiming sender of a security token.

Integrity - *Integrity* is the process by which it is guaranteed that information is not modified in transit.

Confidentiality - *Confidentiality* is the process by which data is protected such that only authorized actors or security token owners can view the data

Digest - A *diges*t 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.

3. Quality of Protection

In order to secure a SOAP message, two types of threats should be considered: 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 we define a message security model.

3.1. Message Security Model

In this document we specify an abstract *message security model* in terms of security tokens combined with digital signatures as proof of possession of the security token (key).

Security tokens assert claims and signatures provide a mechanism for proving the senderâ€™s knowledge of the key. As well, the signature can be used to "bind" or "associate" the signature with the claims in the security token (assuming the token is trusted).

Note that such a binding is limited to those elements covered by the signature. Furthermore note that this document does not specify a particular method for authentication, it simply indicates that security tokens MAY be bound to messages.

A claim can be either endorsed or unendorsed by a trusted authority. A set of endorsed 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 endorsed claim can also be represented as a reference to an authority so that the receiver can "pull" the claim from the referenced authority.

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

One special type of unendorsed claim is Proof-of-Possession. Such a claim proves that the sender has a particular piece of knowledge that is verifiable by, appropriate actors. For example, a username/password is a security token with this type of claim. A Proof-of-Possessionclaim is sometimes combined with other security tokens to prove the claims of the sender. Note that a digital integrity used for message integrity can also be used as a Proof-of-Possession claim, although in this specification we do 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 intercepted (confidentiality) or illegally modified (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 actors, and to be extensible to support additional signature formats.

Message confidentiality leveragesXML 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 actors.

3.3. Missing or Inappropriate Claims

The message receiver SHOULD reject a message with invalid signature, missing or inappropriate claims as it is an unauthorized (or malformed) message. This specification provides a flexible way for the message sender to claim 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.

4. Security Element

The <Security> header block provides a mechanism for attaching security-related information targeted at a specific receiver (SOAP actor). This MAY be either the ultimate receiver 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 <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 <Security> header blocks if they are targeted for separate receivers. However, only one <Security> header block can omit the S:actor attribute and no two <Security> header blocks can have the same value for S:actor. Message security information targeted for different receivers MUST appear in different <Security> header blocks. The <Security> header block without a specified S:actor can be consumed by anyone, but MUST NOT be removed prior to the final destination as determined by WS-Routing.

As elements are added to the <Security> header block, they should be prepended to the existing elements. As such, the <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 <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 subsequent 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>
            ...
        <Security S:actor="..." S:mustUnderstand="...">
            ...
        </Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

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

/Security

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

/Security/@S:actor

This attribute allows a specific SOAP actor to be identified. This attribute is not required; however, no two instances of the header block may omit an actor or specify the same actor.

/Security/{any}

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

/Security/@{any}

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

The following sub-sections outline new and existing elements that are expected to be used within the <Security> header.

4.1. UsernameToken Element

We introduce the <UsernameToken> as a way of proving a username and optional password information.

The following illustrates the syntax of this element:

```
<UsernameToken Id="...">
    <Username>...</Username>
    <Password Type="...">...</Password>
</UsernameToken>
```

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

/UsernameToken

This element is used for sending basic authentication information.

/UsernameToken/@Id

A string label for this security token.

/UsernameToken/Username

This required element specifies the username of the authenticating party.

/UsernameToken/Username/@{any}

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

/UsernameToken/Password

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

/UsernameToken/Password/@Type

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

Value	Description
wsse:PasswordText (default)	The actual password for the username.
wsse:PasswordDigest	The digest of the password for the username. The value is a base64-encoded SHA1 hash value of the UTF8-encoded password.

/UsernameToken/Password/@{any}

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

/UsernameToken/{any}

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

/UsernameToken/@{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:

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Zoe</wsse:Username>
                <wsse:Password>ILoveDogs</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

4.2. Encoding Binary Security Tokens

Any XML-based security token can be specified in the <Security> header. However, binary (e.g., X.509 certificates and Kerberos tickets) or other non-XML formats require a special encoding format for inclusion.

A binary security token has 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 BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute, and the value type and space are specified using the ValueType attribute.

The following is an overview of the syntax:

```
<BinarySecurityToken Id=...
                        EncodingType=...
                        ValueType=.../>
```

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

/BinarySecurityToken

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

/BinarySecurityToken/@Id

An optional string label for this security token.

/BinarySecurityToken/@ValueType

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

/BinarySecurityToken/@EncodingType

The EncodingType attribute is used to indicate, using a QName, the encoding format of the binary data (e.g., wsse:Base64Binary). We introduce this new attribute, 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:

QName	Description
wsse:Base64Binary	XML Schema base 64 encoding
wsse:HexBinary	XML Schema hex encoding

/BinarySecurityToken/@{any}

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

The following value spaces are defined for @ValueType:

QName	Description
wsse:X509v3	X.509 v3 certificate
wsse:Kerberosv5TGT	Kerberos v5 ticket as defined in Section 5.3.1 of Kerberos. This ValueType is used when the ticket is a ticket granting ticket (TGT)
wsse:Kerberosv5ST	Kerberos v5 ticket as defined in Section 5.3.1 of Kerberos. This ValueType is used when the ticket is a service ticket (ST

Note that XML Signature also provides mechanisms for encoding X.509 certificates. The BinarySecurityToken with ValueType="wsse:X509v3" MAY be used when flexibility is required for encoding purposes. On the other hand, using ds:KeyInfo may provide additional flexibility in usage scenarios.

The following example illustrates the use of BinarySecurityToken:

```
  <wsse:BinarySecurityToken
             xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
             Id="myToken"
             ValueType="wsse:X509v3"
             EncodingType="wsse:Base64Binary">
             MIIEZzCCA9CgAwIBAgIQEmtJZc0...
          </wsse:BinarySecurityToken>
```

When a <BinarySecurityToken> is used 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 <BinarySecurityToken> element if this token does not carry the signing 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 <BinarySecurityToken> element. Note that the definition of wsse is also included as it is used for the encoding type and the element.

```
 <wsse:BinarySecurityToken
             xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
             Id="myToken"
             ValueType="x:MyType" xmlns:x="http://fabrikam123.com/x"
             EncodingType="wsse:Base64Binary">
             MIIEZzCCA9CgAwIBAgIQEmtJZc0...
          </wsse:BinarySecurityToken>
```

When a Kerberos ticket is referenced as a signature key, the signature algorithm SHOULD be a hashed message authentication code. In particular, it is RECOMMENDED to use HMAC-SHA1 (required by XML Signature), with the session key in the ticket used as the shared secret key.

4.3. 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 <SecurityTokenReference> element provides an extensible mechanism for referencing security tokens.

The following illustrates the syntax of this element:

```
<SecurityTokenReference Id="...">
    <Reference URI="..."/>
</SecurityTokenReference>
```

The following describes the elements defined above:

/SecurityTokenReference

This element provides a reference to a security token.

/SecurityTokenReference/@Id

A string label for this security token reference

/SecurityTokenReference/Reference

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

/SecurityTokenReference/Reference/@URI

This attribute specifies a URI for where to find a security token.

/SecurityTokenReference/{any}

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

/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:

```
<wsse:SecurityTokenReference
          xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
   <wsse:Reference
             URI="http://www.fabrikam123.com/tokens/Zoe#X509token"/>
</wsse:SecurityTokenReference>
```

This element can also be used as a direct child element of <ds:KeyInfo> to indicate the 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 <SecurityTokenReference> element be placed inside a <ds:KeyInfo> to reference the security token used for the signature or encryption.

4.4. ds:KeyInfo

For certain key types, such as X.509 certificate, both the <ds:KeyInfo> element (from XML Signature) and the

<BinarySecurityToken> element can be used for carrying the key information. The <ds:KeyInfo> element is allowed for different key types and for future extensibility. However, in this specification, the use of <BinarySecurityToken> is the RECOMMENDED way to carry key material if the key type is well defined in Section 4.2.

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

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

4.5. ds:Signature

Message senders may want to enable message receivers 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.

When an XML Signature is used in conjunction with the <SecurityTokenReference> element, the security token of a message signer may be correlated and a mapping made between the claims of the security token and the message as evaluated by the application.

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

All compliant implementations MUST be able to process a <ds:Signature> element

4.5.1. Algorithms

The WS-Security 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 WS-Security RECOMMENDS:

Algorithm Type	Algorithm	Algorithm URI
Canonicalization	Exclusive XML Canonicalization	http://www.w3.org/2001/10/xml-exc-c14n
Transformations	XML Decryption Transformation	http://www.w3.org/2001/04/decrypt

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

Finally, if a sender wishes to sign a message before encryption, they should use the Decryption Transformation for XML Signature.

4.5.2. Signing Messages

The <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 <Security> header block. Senders should take care to sign all important elements of the message, but care must be taken in creating a 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 <Security> header block, a <ds:Signature> element conforming to the XML Signature specification SHOULD be prepended to the existing content of the <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.

4.5.3. Verifying Integrity

The validation of a <ds:Signature> entry inside an <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 trust 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 6.

4.5.4. Example

The following sample message illustrates the use of integrity and security tokens. For this example, we use a fictitious "RoutingTransform" that selects the immutable routing headers along with the message body.

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp">
         <m:action>http://fabrikam123.com/getQuote</m:action>
         <m:to>http://fabrikam123.com/stocks</m:to>
         <m:from>mailto:johnsmith@fabrikam123.com</m:from>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
      <wsse:Security>
         <wsse:BinarySecurityToken
                     ValueType="wsse:X509v3"
                     EncodingType="wsse:Base64Binary"
                     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>
                  <ds:Transforms>
                     <ds:Transform Algorithm=
                           "http://...#RoutingTransform"/>
                     <ds:Transform Algorithm=
                           "http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </ds:Transforms>
                  <ds:DigestMethod Algorithm=
                       "http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <ds:DigestValue>EULddytSo1...</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>
              BL8jdfToEb1l/vXcMZNNjPOV...
            </ds:SignatureValue>
            <ds:KeyInfo>
                <wsse:SecurityTokenReference>
                    <wsse:Reference URI="#X509Token"/>
                </wsse:SecurityTokenReference>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
   </S:Header>
   <S:Body>
      <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
        QQQ
      </tru:StockSymbol>
   </S:Body>
</S:Envelope>
```

4.6. Encryption Sub-elements

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 receiver or a key carried in the message in an encrypted form.

In order to allow this flexibility, we leverage the XML Encryption standard. Specifically, we describe how three elements (listed below and defined in XML Encryption) can be used within the <Security> header block. When a sender or an intermediary encrypts portion(s) of a SOAP message using XML Encryption they will add a sub-element to the <Security> header block. Furthermore, the encrypting party MUST prepend the sub-element into the <Security> header block for the targeted receiver 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 receiver to identify which portions of the message are to be decrypted by the receiver.

4.6.1. 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 receiver use a shared secret key. The following illustrates the use of this sub-element:

```
<S:Envelope
   xmlns:S="http://www.w3.org/2001/12/soap-envelope"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
        <wsse:Security>
            <xenc:ReferenceList>
                <xenc:DataReference URI="#bodyID"/>
            </xenc:ReferenceList>
        </wsse:Security>
    </S:Header>
    <S:Body>
        <xenc:EncryptedData Id="bodyID">
          <ds:KeyInfo>
<ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
</ds:KeyInfo>
<xenc:CipherData>
   <xenc:CipherValue>...</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
    </S:Body>
</S:Envelope>
```

4.6.2. 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 recipientâ€™s key and embedded in the message, <xenc:EncryptedKey> MAY be used for carrying such an encrypted key. This sub-element SHOULD have a manifest, that is, an <xenc:ReferenceList> element, in order for the recipient to know the portions to be decrypted with this key (if any exist). An element or element content to be encrypted by this encryption step MUST be replaced by a corresponding <xenc:EncryptedData> according to XML Encryption. All the <xenc:EncryptedData> elements created by this encryption step SHOULD be listed in the <xenc:ReferenceList> element inside this sub-element.

This construct is useful when encryption is done by a randomly generated symmetric key that is in turn encrypted by the recipientâ€™s public key. The following illustrates the use of this element:

```
<S:Envelope
   xmlns:S="http://www.w3.org/2001/12/soap-envelope"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
        <wsse:Security>
            <xenc:EncryptedKey>
               <xenc:EncryptionMethod Algorithm="..."/>
               <ds:KeyInfo>
                   <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
               </ds:KeyInfo>
               <xenc:CipherData>
                   <xenc:CipherValue>...</xenc:CipherValue>
               </xenc:CipherData>
               <xenc:ReferenceList>
                  <xenc:DataReference URI="#bodyID"/>
               </xenc:ReferenceList>
            </xenc:EncryptedKey>
        </wsse:Security>
    </S:Header>
    <S:Body>
        <xenc:EncryptedData Id="bodyID">
            <ds:KeyInfo>
   <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
  </ds:KeyInfo>
<xenc:CipherData>
    <xenc:CipherValue>...</xenc:CipherValue>
            </xenc:CipherData>
</xenc:EncryptedData>
    </S:Body>
</S:Envelope>
```

4.6.3. 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:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <S:Header>
        <wsse:Security>
            <xenc:EncryptedData MimeType="image/png">
               <xenc:EncryptionMethod Algorithm="foo:bar"/>
               <ds:KeyInfo>
   <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
     </ds:KeyInfo>
               <xenc:CipherData>
                   <xenc:CipherReference URI="cid:image"/>
               </xenc:CipherData>
            </xenc:EncryptedData>
        </wsse:Security>
    </S:Header>
    <S:Body> </S:Body>
</S:Envelope>
```

4.6.4. 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 toXML 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 replaces the <xenc:EncryptedData>. However, if the <enc:EncryptedData> element is located in the <Security> header block and it refers to an attachment, then the decrypted octet stream MUST replace the encrypted octet stream in the attachment.

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 "actor" 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 4.5.3.
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>.

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 6.

5. Extended Example

The following sample message illustrates the use of security tokens, signatures, and encryption. For this example, we use a fictitious "RoutingTransform" that selects the immutable routing headers along with the message body.

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
```

```
<S:Header>
   <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
      <m:action>http://fabrikam123.com/getQuote</m:action>
      <m:to>http://fabrikam123.com/stocks</m:to>
      <m:from>mailto:johnsmith@fabrikam123.com</m:from>
      <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
   </m:path>
   <wsse:Security>
      <wsse:BinarySecurityToken
             ValueType="wsse:X509v3"
             Id="X509Token"
             EncodingType="wsse:Base64Binary">
      MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>
      <xenc:EncryptedKey>
          <xenc:EncryptionMethod Algorithm=
                "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
          <ds:KeyInfo>
            <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
          </ds:KeyInfo>
          <xenc:CipherData>
             <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
             </xenc:CipherValue>
          </xenc:CipherData>
          <xenc:ReferenceList>
              <xenc:DataReference URI="#enc1"/>
          </xenc:ReferenceList>
      </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>
               <ds:Transforms>
                  <ds:Transform
                     Algorithm="http://...#RoutingTransform"/>
                  <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>LyLsF094hPi4wPU...
                </ds:DigestValue>
            </ds:Reference>
         </ds:SignedInfo>
         <ds:SignatureValue>
                  Hp1ZkmFZ/2kQLXDJbchm5gK...
         </ds:SignatureValue>
         <ds:KeyInfo>
             <wsse:SecurityTokenReference>
                 <wsse:Reference URI="#X509Token"/>
             </wsse:SecurityTokenReference>
         </ds:KeyInfo>
      </ds:Signature>
   </wsse:Security>
```

```
(051)   </S:Header>
(052)   <S:Body>
(053)      <xenc:EncryptedData
                  Type="http://www.w3.org/2001/04/xmlenc#Element"
                  Id="enc1">
(054)         <xenc:EncryptionMethod
              Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
(055)         <xenc:CipherData>
(056)            <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(057)            </xenc:CipherValue>
(058)         </xenc:CipherData>
(059)      </xenc:EncryptedData>
(060)   </S:Body>
(061) </S:Envelope>
```

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

Lines (003)-(051) contain the SOAP message headers.

Lines (004)-(009) specify the message routing information (as define in WS-Routing). In this case we are sending the message to the http://fabrikam123.com/stocks service requesting the "getQuote" action.

Lines (010)-(050) represent the <Security> header block. This contains the security-related information for the message.

Lines (011)-(013) 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 (012) specifies the actual Base64 encoding of the certificate.

Lines (014)-(026) specify the key that is used to encrypt the body of the message. Since this is a symmetric key, it is passed in an encrypted form. Line (015) defines the algorithm used to encrypt the key. Lines (016)-(018) specify the name of the key that was used to encrypt the symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines (023)-(025) identify the encryption block in the message that uses this symmetric key. In this case it is only used to encrypt the body (Id="enc1").

Lines (027)-(049) specify the digital signature. In this example, the signature is based on the X.509 certificate. Lines (028)-(040) indicate what is being signed. Specifically, Line (029) indicates the canonicalization algorithm (exclusive in this example). Line (030) indicate the signature algorithm (rsa over sha1 in this case).

Lines (031)-(039) identify the parts of the message that are being signed. Specifically, Line (033) identifies a "transform". This fictitious transforms selects the immutable portions of the routing header and the message body. Line (034) specifies the canonicalization algorithm to use on the selected message parts from line (033). Line (036) indicates the digest algorithm use on the canonicalized data. Line (037) specifies the digest value resulting from the specified algorithm on the canonicalized data.

Lines (041)-(043) indicate the actual signature value - specified in Line (042).

Lines (044)-(048) indicate the key that was used for the signature. In this case, it is the X.509 certificate included in the message. Line (046) provides a URI link to the Lines (011)-(013).

The body of the message is represented by Lines (052)-(060).

Lines (053)-(059) represent the encrypted metadata and form of the body using XML Encryption. Line (053) indicates that the "element value" is being replaced and identifies this encryption. Line (054) specifies the encryption algorithm - Triple-DES in this case. Lines (055)-(058) contain the actual cipher text (i.e., the result of the encryption). Note that we don't include a reference to the key as the key references this encryption - Line (024).

6. 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 receiver MAY provide a response that informs the sender of supported formats, etc. For failure errors, the receiver MAY choose not to respond, as this may be a form of Denial of Service (DOS) or cryptographic attack. We combine signature and encryption failures to mitigate certain types of attacks.

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

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

The "failure" class of errors are:

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

7. Security Considerations

It is strongly RECOMMENDED that messages include digitally signed elements to allow message receivers 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. Care should be taken by application designers not to introduce such vulnerabilities.

8. Acknowledgements

This specification has been developed as a result of joint work with many individuals and teams, including:

Bob Blakley, IBM

Allen Brown, Microsoft

Kelvin Lawrence, IBM

Scott Konersmann, Microsoft

David Melgar, IBM

9. References

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

[Kerberos]
J. Kohl and C. Neuman, "*The Kerberos Network Authentication Service(V5)*," RFC 1510 , September 1993.

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

[SHA-1]
FIPS PUB 180-1. *Secure Hash Standard. U.S. Department of Commerce / National Institute of Standards and Technology*.

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

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

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

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

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

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

[XML-Schema1]
W3C Recommendation, "*XML Schema Part 1: Structures* ,"2 May 2001.

[XML-Schema2]
W3C Recommendation, "*XML Schema Part 2: Datatypes*," 2 May 2001.

[XML Signature]
W3C Proposed Recommendation, "*XML Signature Syntax and Processing*," 20 August 2001.

[WS-Routing]

H. Nielsen, S. Thatte, "*Web Services Routing Protocol*", Microsoft, October 2001.

[X509]

S. Santesson, et al,"*Internet X.509 Public Key Infrastructure Qualified Certificates Profile*,"

[XPath]

W3C Recommendation, "*XML Path Language*," 16 November 1999.