WS-Security AppNotes

e-mail it!

A joint WS-Security Application Note from IBM Corporation and Microsoft Corporation.

August 2002

This is a preliminary document and may be changed substantially over time. The information contained in this document represents the current view of International Business Machine and Microsoft Corporation on the issues discussed as of the date of publication. Because IBM and Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of IBM and Microsoft, and IBM and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

The presentation, distribution or other dissemination of the information contained in this document is not a license, either expressly or impliedly, to any intellectual property owned or controlled by IBM or Microsoft and\or any other third party. IBM, Microsoft and\or any other third party may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to IBM's or Microsoft's or any other third party's patents, trademarks, copyrights, or other intellectual property. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

This document and the information contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, IBM and Microsoft provides the document AS IS AND WITH ALL FAULTS, and hereby disclaims all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence, all with regard to the document. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE DOCUMENT.

IN NO EVENT WILL IBM OR MICROSOFT BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THIS DOCUMENT, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Related content:

Subscribe to the developerWorks newsletter

Also in the Web services zone:

Tutorials
Tools and products
Articles

Audience

This paper is provided as guidance to implementers of the WS-Security [WSSEC] specification. This application note applies to both WS-Security and the associated addendum [WSSECA]. Consequently, the discussions here apply to the schemas identified in both specifications.

Introduction

The Web Services Security specification (WS-Security) provides a set of mechanisms to help developers of Web Services secure SOAP message exchanges. Specifically, WS-Security describes enhancements to the existing SOAP messaging to provide *quality of protection* through the application of message integrity, message confidentiality, and single message authentication to SOAP

messages. These basic mechanisms can be combined in various ways to accommodate building a wide variety of security models using a variety of cryptographic technologies.

WS-Security also provides a general-purpose mechanism for associating security tokens with messages. However, no specific type of security token is required by WS-Security. It is designed to be extensible (e.g. support multiple security token formats) to accommodate a variety of authentication and authorization mechanisms. For example, a requestor might provide proof of identity and a signed claim that they have a particular business certification. A Web service, receiving such a message could then determine what kind of trust they place in the claim.

Additionally, WS-Security describes how to encode binary security tokens and attach them to SOAP messages. Specifically, the WS-Security specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys as a sample of different binary token types. Kerberos tickets and X509 certificates are used today by developers to add authentication mechanisms to many Web applications. With WS-Security, the domain of these mechanisms can be extended by carrying authentication information in Web services requests. WS-Security also includes extensibility mechanisms that can be used to further describe the credentials that are included with a message. WS-Security is a building block that can be used in conjunction with other Web service protocols to address a wide variety of application security requirements.

Message integrity is provided by leveraging XML Signature [DIGSIG] and security tokens to ensure that messages have originated from the appropriate sender and were not modified in transit. Similarly, message confidentiality leverages XML Encryption and security tokens to keep portions of a SOAP message confidential.

The SOAP Composable Architecture and WS-Security
The SOAP messaging model is based on an extensible framework and SOAP-based specifications are designed to be composed with each other to provide a rich messaging environment. As such, WS-Security by itself does not provide a complete security solution but it can be used in conjunction with other Web service and application specific protocols to accommodate a wide variety of security models and encryption technologies.

Terminology
Here we reiterate some basic definitions for the security terminology used in the WS-Security specification.

Claim - A *claim* is a statement that a requestor 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).

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 *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

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.

Message Security Background
Security tokens assert claims which can be coupled with digital signatures to provide mechanisms for demonstrating evidence of the sender's knowledge of the keys described by the security token. In addition the definition of a SOAP header element provides a mechanism for "associating" the signature with the claims in the security token.

Several caveats to note: First, note that such a binding is limited to those elements covered by the signature. Second, note that these mechanisms do not specify a particular method for authentication; it simply indicates that security tokens may be bound to

messages. Third, note that the message recipient may or may not trust the security tokens. Finally, note that these security model mechanisms are not a complete solution by themselves; additional specifications are required for a complete security solution.

Security token claims can be either endorsed by an authority or be left unendorsed. A set of endorsed claims is usually represented as a signed security token that is digitally signed or cryptographically protected by the authority. An X.509 certificate, which asserts a binding between one's identity and public key, is an example of a signed security token. Security tokens can be "pushed" or carried in a message, or the security token can be a reference that the receiver can use to "pull" the claim from the referenced authority.

Another aspect of a security model is the articulation of a trust relationship. Security tokens are useful within a trust domain. A trust domain can be articulated through a manual process, an agreement or an implementation of a set of rules enforcing the trust policy. An unendorsed claim can thus be trusted if there is any established 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 connection with a sufficient protection and there is an out-of-band trust relationship between them.

One special type of unendorsed claim is Proof-of-Possession. Such a claim produces evidence that the sender has a particular piece of knowledge that is only known to, or verifiable by, appropriate actors [SOAP]. For example, a username/password is a security token with this type of claim. In this type of scenario, the receiver makes a decision about whether or not the evidence produced is sufficient proof. This type of evidence is sometimes combined with other security tokens to prove the claims of the sender. Note that a digital signature applied to a message for the purposes of message integrity can also be interpreted as evidence of possession of part of the key pair although in this specification we do not consider such a digital signature as a type of security token.

Protecting the message content from being illegally accessed (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 leverages XML Encryption [XML-Encrypt] 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.

WS-Security 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">
(003)   <S:Header>
(004)     <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
(005)       <m:action>http://fabrikam123.com/getQuote</m:action>
(006)       <m:to>http://fabrikam123.com/stocks</m:to>
(007)       <m:from>mailto:johnsmith@fabrikam123.com</m:from>
(008)       <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
(009)     </m:path>
(010)     <wsse:Security
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
                        xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
(011)       <wsse:BinarySecurityToken
              ValueType="wsse:X509v3"
              wsu:Id="X509Token"
              EncodingType="wsse:Base64Binary">
(012)           MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
(013)       </wsse:BinarySecurityToken>
(014)       <xenc:EncryptedKey
              xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
(015)           <xenc:EncryptionMethod
                  Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
```

```
(016)             <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
(017)               <wsse:SecurityTokenReference>
                        <wsse:KeyIdentifier>FHejk...
                        </wsse:KeyIdentifier>
                    </wsse:SecurityTokenReference>
(018)             </ds:KeyInfo>
(019)             <xenc:CipherData>
(020)                <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
(021)                </xenc:CipherValue>
(022)             </xenc:CipherData>
(023)             <xenc:ReferenceList>
(024)                 <xenc:DataReference URI="#enc1"/>
(025)             </xenc:ReferenceList>
(026)          </xenc:EncryptedKey>
(027)          <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
(028)             <ds:SignedInfo>
(029)               <ds:CanonicalizationMethod
                      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(030)               <ds:SignatureMethod
                      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(031)               <ds:Reference>
(032)                  <ds:Transforms>
(033)                     <ds:Transform
                            Algorithm="http://...#RoutingTransform"/>
(034)                     <ds:Transform
                            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(035)                  </ds:Transforms>
(036)                  <ds:DigestMethod
                         Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(037)                  <ds:DigestValue>LyLsF094hPi4wPU...
(038)                   </ds:DigestValue>
(039)               </ds:Reference>
(040)             </ds:SignedInfo>
(041)             <ds:SignatureValue>
(042)                    Hp1ZkmFZ/2kQLXDJbchm5gK...
(043)             </ds:SignatureValue>
(044)             <ds:KeyInfo>
(045)                 <wsse:SecurityTokenReference>
(046)                     <wsse:Reference URI="#X509Token"/>
(047)                 </wsse:SecurityTokenReference>
(048)             </ds:KeyInfo>
(049)          </ds:Signature>
(050)       </wsse:Security>
(051)    </S:Header>
(052)    <S:Body>
(053)       <xenc:EncryptedData
              xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
              xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
              Type="http://www.w3.org/2001/04/xmlenc#Element"
              wsu: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 identifier of the key that was used to encrypt the symmetric key. Lines (019)-(022) specify the actual encrypted form of the symmetric key. Lines (023)-(025) identify the encryption block in the message that uses this symmetric key. In this case it is only used to encrypt the body (wsu: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) indicates the signature algorithm (RSA over SHA1 in this case).

Lines (031)-(039) identify the parts of the message that 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 to use on the canonicalized data. Line (037) specifies the digest value resulting from the specified algorithm on the canonicalized data.

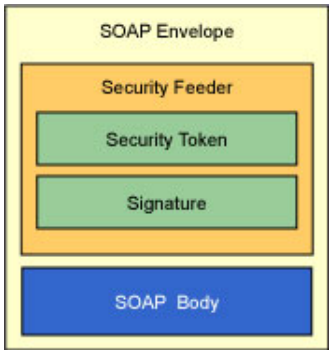
Lines (041)-(043) indicate the actual signature value - specified in Line (042).

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 the CBC mode 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).

A general illustration of the message format is shown below.

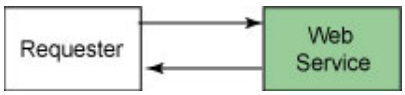


Using WS-Security
Below we present a number of examples that show the use of Web Service security. These examples are designed to illustrate how the mechanisms of WS-Security can be applied to address specific security issues - these examples are not meant to be definitive examples of how to build complete end-to-end secure solutions

The following list briefly describes each of the examples scenarios we describe.

- Direct Trust using Username/Password -This scenario illustrates using a username and password with transport security.
- Direct Trust using Security Tokens - This scenario illustrates direct trust using X.509 certification.
- Security Token Acquisition - This scenario illustrates authentication using a security token stored independently from the message.
- Issued Security Token - This scenario illustrates basic authentication.
- Firewall Processing - This scenario illustrates how firewalls can leverage this security model for greater degrees of control

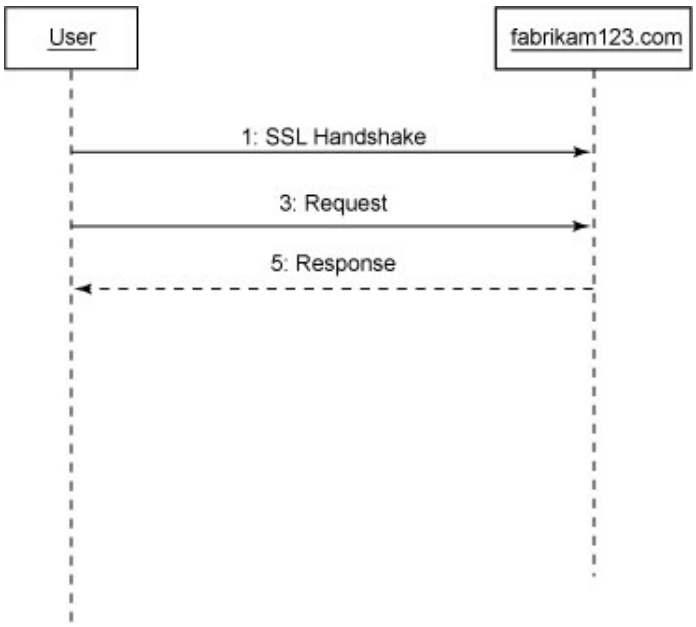
Direct Trust using Username/Password



With SSL/TLS the server hosting the web service uses a public key pair to establish a secure channel with the client over an http connection. Once the two entities have exchanged keys, the server prompts the client for their userid id and password through an http message exchange. The username and password are carried in an http header.

The variation using SOAP messaging below shows how Web services Security can be used with existing transport security.

The client opens a connection to the Web service using a secure transport. It sends its request and includes a security token that contains its username and password. The service authenticates the information, processes the request, and returns the result.



The example would operate as follows:

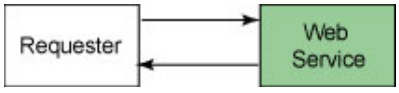
1. The requestor opens a connection to the Web service using a secure transport (which is not persistent) such as SSL (the interactions for this are omitted).
2. The requestor constructs a SOAP message. Included in this is a <UsernameToken> element in the <Security> header block. This element contains the requestor's username and password at the service. The password can be sent in the clear because the transport is secure.
3. The message is sent to the service.
4. The service extracts the <UsernameToken> element and validates the username and password.
5. Since the validation succeeded, the service processes the message and returns a result.

The following shows a sample SOAP envelope that is sent from the requestor to the service.

```
(001) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope">
(002)   <S:Header>
(003)     <wsse:Security
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
            S:mustUnderstand="1">
(004)       <wsse:UsernameToken>
(005)         <wsse:Username>Zoe</wsse:Username>
(006)         <wsse:Password>ILoveDogs</wsse:Password>
(007)       </wsse:UsernameToken>
(008)     </wsse:Security>
(009)   </S:Header>
(010)   <S:Body>
(011)     <m:GetLastTradePrice xmlns:m="http://fabrikam123.com/">
(012)       <symbol>DIS</symbol>
(013)     </m:GetLastTradePrice>
(014)   </S:Body>
(015) </S:Envelope>
```

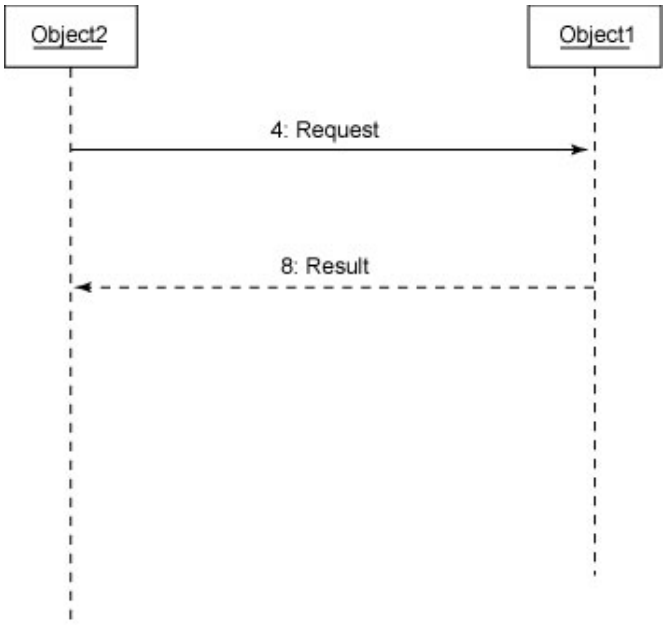
The UsernameToken is also useful when the transport does not provide cryptographic protection against eavesdropping and modification, such as plain TCP/IP, if the network is considered to be secure enough according to its management policy. A company's intranet can be such a trusted network.

Direct Trust using Security Token and Signature



This example illustrates the use of a security token that is directly trusted by a Web service. Here direct trust means that the requestor's security token (or its signing authority) is known and trusted by the Web service. This scenario assumes that the two parties have used some mechanism to establish the Web service trust in the security token. This trust may be established manually, or by configuring the application. No assumption is made about the organizational relationship between the parties.

The requestor sends a message to a service and includes a signed security token and provides proof-of-possession of the key associated with the security token. The service verifies the proof and evaluates the security token. The signature on the security token is valid and is directly trusted by the service. The service processes the request and returns a result.



The example would operate as follows:

1. The requestor constructs a SOAP message for the service.
2. The requestor computes a signature for the key elements of the message (e.g. the immutable elements in the WS-Routing header block as well as the SOAP `<Body>` element) using XML Signature. The digest is signed using the private key associated with the requestor's X.509 certificate. The `<ds:Signature>` element is placed in the `<Security>` header of the message.
3. The requestor prepends the X.509 certificate using the `<BinarySecurityToken>` in the `<Security>` header. For this element, the ValueType attribute is `wsse:x509v3` and the EncodingType is `wsse:base64Binary`
4. The requestor sends the message to the service
5. The service extracts the X.509 certificate from the `<BinarySecurityToken>` element in the `<Security>` header.
6. The service determines that the certificate authority on the certificate is in its "trusted" list and that the authority signature is valid.
7. The service validates the `<ds:Signature>` in the `<Security>` header.
8. The service authenticates the requestor so it processes the message and returns a result. (Note that a similar mechanism can be used to authenticate the service's response to the requestor)

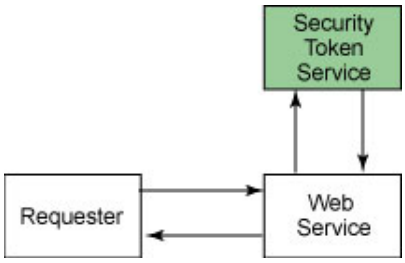
The following shows a sample SOAP envelope that is sent from the requestor to the service.

```
(001) <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
(002)   <S:Header>
(003)     <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
(004)        <m:action>http://fabrikam123.com/getQuote</m:action>
(005)        <m:to>http://fabrikam123.com/stocks</m:to>
(006)        <m:from>mailto:johnsmith@fabrikam123.com</m:from>
(007)        <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
(008)     </m:path>
(009)     <wsse:Security
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
            S:mustUnderstand="1">
(010)       <wsse:BinarySecurityToken
              EncodingType="wsse:Base64Binary"
              wsu:Id="X509Token"
              ValueType="wsse:X509v3">
(011)          MIIDQTCCAqqgAwIBAgICAZIhvcNAQEFBQAwTjELMAkGA1UEBhMCS...
(012)       </wsse:BinarySecurityToken>
(013)       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
(014)         <SignedInfo>
(015)           <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
(016)           <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
(017)           <Reference URI="">
(018)             <Transforms>
(019)               <Transform Algorithm="http://...#RoutingTransform"/>
(020)               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(021)             </Transforms>
(022)             <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(023)             <DigestValue>yH8GsCH3FT747UhqqzHqqSTj7CM=</DigestValue>
(024)           </Reference>
(025)         </SignedInfo>
(026)         <SignatureValue>
(027)          TQtb/T9NTGgRzbtDCctfw0SKY7/PHVZtPMFoLtPCixU3Kobis/Q...
(028)        </SignatureValue>
(029)         <KeyInfo>
(030)           <wsse:SecurityTokenReference>
(031)             <wsse:Reference URI="#X509Token"/>
(032)           </wsse:SecurityTokenReference>
(033)         </KeyInfo>
(034)       </Signature>
(035)     </wsse:Security>
(036)   </S:Header>
```

```
(037)   <S:Body>
(038)     <m:GetLastTradePrice xmlns:m="http://www.fablikam123.com/">
(039)       <symbol>DIS</symbol>
(040)     </m:GetLastTradePrice>
(041)   </S:Body>
(042) </S:Envelope>
```

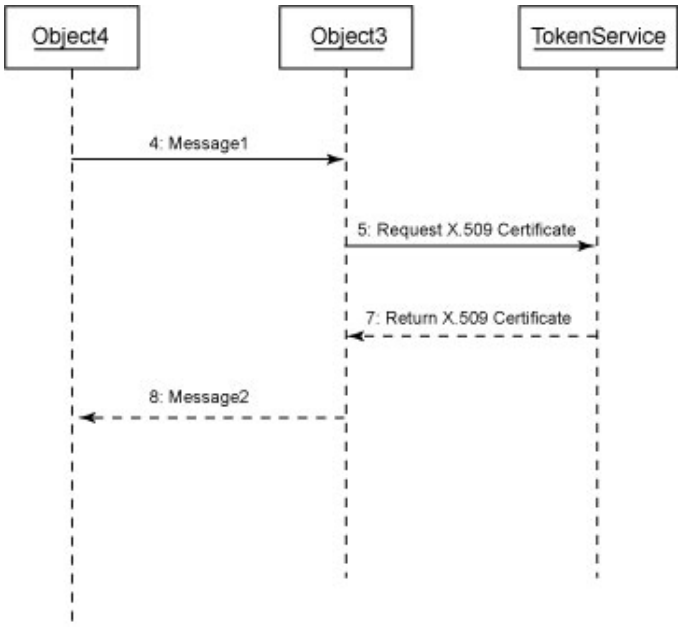
In this sample, lines (010)-(012) indicate the security token used for the signature. Here, we assume an X.509 self-signed certificate that is directly trusted by the service.

Security Token Acquisition



In some cases, the security token used isn't passed as part of the message. Instead, a security token reference is provided that can be used to locate and acquire the token.

The requestor issues a request to the service and includes a reference to the security token and provides proof-of-possession in the form of XML Signature. The Web service uses the provided information to obtain the security token from the token store service and validate the proof. The Web service trusts (note that trust was established outside of the message semantics) the security token, so the request is processed and the response is returned.



The example would operate as follows:

1. The requestor constructs a SOAP message for the service.
2. The requestor computes a signature for the key elements of the message (see previous scenario) using XML Signature. The digest is signed using the private key associated with the requestor's X.509 certificate. The `<ds:Signature>` element is placed in the `<Security>` header block of the message.
3. The requestor's X.509 certificate is at a specific Web location. Consequently, rather than passes it in the message, the requestor prepends a reference to its X.509 certificate using the `<SecurityTokenReference>` element in the `<Security>` header block. In this case, the element contains an `<Reference>` element that specifies the URL for the

certificate.

4. The requestor sends the message to the service.
5. The service extracts the reference to the X.509 certificate from the `<SecurityTokenReference>` element in the `<Security>` header block.
6. The service fetches the X.509 certificate from the specified URL (security token service).
7. The URL (security token service) returns the requested certificate.
8. The service determines that the certificate authority on the certificate is in its "trusted" list and that the authority signature is valid.
9. The service validates the `<ds:Signature>` in the `<Security>` header.
10. The service authenticates the requestor so it processes the message and returns a result. (Note that a similar mechanism can be used to authenticate the service's response to the requestor)

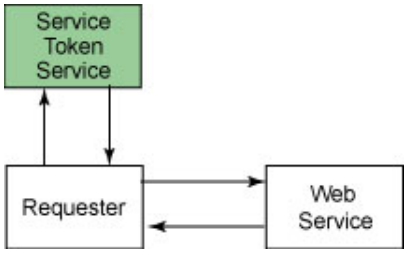
The following shows a sample SOAP envelope that is sent from the requestor to the service (message 1 in the figure above).

```
(001) <S:Envelope
        xmlns:S="http://www.w3.org/2001/12/soap-envelope">
(002)   <S:Header>
(003)     <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
(004)        <m:action>http://fabrikam123.com/getQuote</m:action>
(005)        <m:to>http://fabrikam123.com/stocks</m:to>
(006)        <m:from>mailto:johnsmith@fabrikam123.com</m:from>
(007)        <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
(008)     </m:path>
(009)     <wsse:Security
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
            S:mustUnderstand="1">
(010)       <wsse:SecurityTokenReference wsu:Id="token1">
(011)          <wsse:Reference URI="ldap://fabrikam123.com/CN=John..."/>
(012)       </wsse:SecurityTokenReference>
(013)       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
(014)         <SignedInfo>
(015)           <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
(016)           <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
(017)           <Reference URI="">
(018)             <Transforms>
(019)               <Transform
Algorithm="http://...#RoutingTransform"/>
(020)               <Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(021)             </Transforms>
(022)             <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(023)             <DigestValue>yH8G3FT747UhqqzHqqSTj7CM=</DigestValue>
(024)           </Reference>
(025)         </SignedInfo>
(026)         <SignatureValue>
(027)           TQtb/+yR56T9NTGgRzbtDCctfw0SKY7/Pq2FoLtPCixU3Kobis/Q...
(028)        </SignatureValue>
(029)         <KeyInfo>
(030)           <wsse:SecurityTokenReference>
(031)             <wsse:Reference URI="#token1"/>
(032)           </wsse:SecurityTokenReference>
(033)         </KeyInfo>
(034)       </Signature>
(035)     </wsse:Security>
(036)   </S:Header>
(037)   <S:Body>
(038)     <m:GetLastTradePrice xmlns:m="http://www.foo.com/">
(039)       <symbol>DIS</symbol>
(040)     </m:GetLastTradePrice>
```

```
(041)   </S:Body>
(042) </S:Envelope>
```

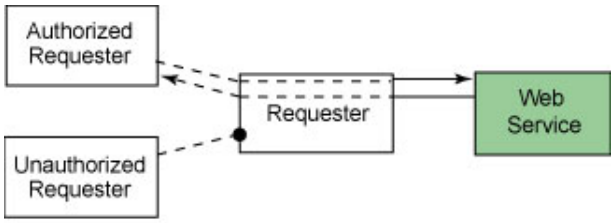
In the above example, lines (010)-(012) indicates a the reference to the security token used in the signature element in lines (013)-(034). This security token reference is in turn referenced by the `<KeyInfo>` element in line (031). This double indirection is recommended because the receiving party can retrieve and validate the security token before processing the `<Signature>` element.

Issued Security Token



This scenario is similar to the previous one but instead of acquiring a security token by the service, a security token is issued to the requester prior to making the request, and the token is carried with the request message using a `<BinarySecurityToken>` element. The security token could be a long-lived token such as an X509 certificate valid for one year, or a short-lived one such as a Kerberos ticket that has a much shorter validity duration. In either case, once a security token is issued by the security token service, the requester may use it for multiple times, sometimes for different Web services if the issued token allows such usage.

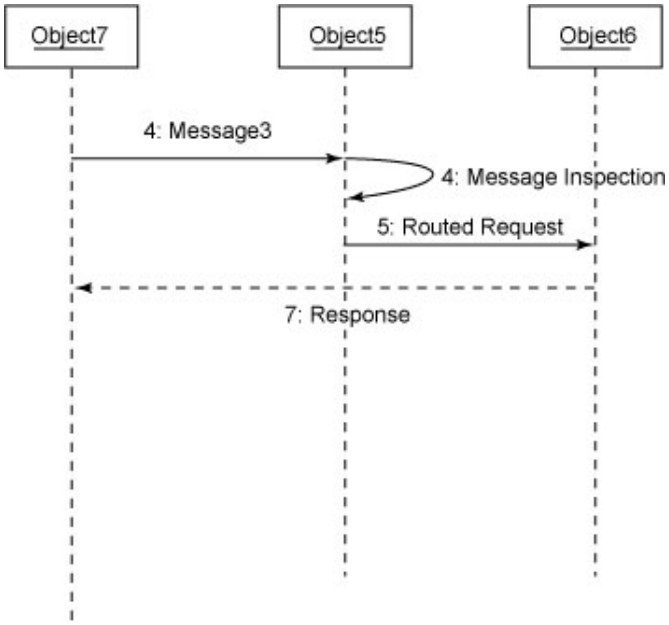
Issued Security Token



Firewalls remain a critical component of the Web services security architecture - they must be able to continue to enforce boundary processing rules.

As shown above, the firewall can examine incoming SOAP messages and only allows those from "authorized" requestors to penetrate the firewall. In this example the firewall examines the messages to determine if the requestor is "authorized" to send messages to the specified Web service inside the firewall.

In this scenario, the firewall can make this decision by examining the `<Security>` header block specifically targeted to the firewall using the actor attribute. This header block contains a security token and a signature on the message. If the signature is valid, and the signing authority for the security token is trusted to authorize messages into the firewall, and the token says that it does authorize messages into the firewall, then the message is allowed; otherwise it is rejected. In some cases, the firewall may examine the `<Security>` header for the ultimate destination instead of a header block explicitly targeted for the firewall.



This scenario would operate as follows:

1. The sender constructs a SOAP message to send to a Web service. The sender may, or may not, include a signature on the message.
2. After providing end-to-end security by signing and encrypting the message, the sender adds an additional `<Security>` header block with the firewall listed as the SOAP actor. Into this header block the requestor places a `<ds:Signature>` element containing a signature over the encrypted data. As well, the security token (or reference) used for the signature is prepended, possibly using `<BinarySecurityToken>`. The sender also specifies the routing information in the Routing header block.
3. The message is sent to the firewall by a transport protocol. The firewall locates the `<Security>` header block targeted for itself and extracts the security token and signature.
4. The firewall validates the signature over the encrypted data and the validity of the security token included (or referenced). The firewall then makes a determination, possibly using external data, as to whether or not to authorize the message to pass through the firewall. If the message is to be allowed to be delivered through the firewall, the firewall removes the `<Security>` header block targeted to itself and modifies the return path information on the Routing header block.
5. The firewall routes the modified message to the ultimate destination using a separate transport.
6. The service validates the `<Security>` header block targeted for it and authenticates the message.
7. The service processes the request and returns a reply.

The following shows a sample SOAP envelope that is sent from the requestor to the firewall. Note that it contains two `<wsse:Security>` header blocks, one is targeted for the firewall with the actor attribute and the other for the ultimate receiver, which is signified by the absence of the actor attribute.

```
(001) <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
(002)   <S:Header>
(003)     <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
(004)       <m:action>http://fabrikam123.com/getQuote</m:action>
(005)       <m:to>http://fabrikam123.com/stocks</m:to>
(006)       <m:fwd>
(007)         <m:via>http://firewall.fabrikam123.com/</m:via>
(008)       </m:fwd>
(009)       <m:from>mailto:johnsmith@fabrikam123.com</m:from>
(010)       <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
(011)     </m:path>
(012)     <wsse:Security
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
            S:actor="http://firewall.fabrikam123.com/"
            S:mustUnderstand="1">
(013)       <wsse:BinarySecurityToken
              EncodingType="wsse:Base64Binary"
```

```
              wsu:Id="X509Token4Firewall"
              ValueType="wsse:X509v3">
(014)          MIIDQTCCAqqgAwIBAgICAQQwDQYJ...
(015)       </wsse:BinarySecurityToken>
(016)       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
(017)         <SignedInfo>
(018)           <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
(019)           <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
(020)           <Reference URI="">
(021)             <Transforms>
(022)               <Transform Algorithm="http://...#RoutingTransform"/>
(023)               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(024)             </Transforms>
(025)             <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(026)             <DigestValue>yH8GsCH3FTqzHqqSTj7CM=</DigestValue>
(027)           </Reference>
(028)         </SignedInfo>
(029)         <SignatureValue>
(030)           f5JJPvwToxNBBzy5R3om/ZtTK63jw...
(031)         </SignatureValue>
(032)         <KeyInfo>
(033)           <wsse:SecurityTokenReference>
(034)             <wsse:Reference URI="#X509Token4Firewall"/>
(035)           </wsse:SecurityTokenReference>
(036)         </KeyInfo>
(037)       </Signature>
(038)     </wsse:Security>

(039)     <wsse:Security
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
            S:mustUnderstand="1">
(040)       <wsse:BinarySecurityToken
              EncodingType="wsse:Base64Binary"
              wsu:Id="X509Token4Stockquote"
              ValueType="wsse:X509v3">
(041)          MIIDQTCCAqqgAwIBAgICAQQwDQYJKoZIhvcNAQEF...
(042)       </wsse:BinarySecurityToken>
(043)       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
(044)         <SignedInfo>
(045)           <CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(046)           <SignatureMethod
 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
(047)           <Reference URI="">
(048)             <Transforms>
(049)               <Transform
 Algorithm="http://...#RoutingTransform"/>
(050)               <Transform
 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(051)             </Transforms>
(052)             <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(053)             <DigestValue>yH8GsCH3FT7qqSTj7CM=</DigestValue>
(054)           </Reference>
(055)         </SignedInfo>
(056)         <SignatureValue>
(057)           9OZ21MKnCQi2Jglw2qIO5e1azezl/j0BP3h2Ut...
(058)         </SignatureValue>
(059)         <KeyInfo>
(060)           <wsse:SecurityTokenReference>
(061)             <wsse:Reference URI="#X509Token4Stockquote"/>
```

```
(062)           </wsse:SecurityTokenReference>
(063)         </KeyInfo>
(064)       </Signature>
(065)     </wsse:Security>
(066)   </S:Header>
(067)   <S:Body>
(068)     <m:GetLastTradePrice xmlns:m="http://www.fabrikam123.com/">
(069)       <symbol>DIS</symbol>
(070)     </m:GetLastTradePrice>
(071)   </S:Body>
(072) </S:Envelope>
```

After authorized by the firewall, the message is modified (the routing header is changed as well as the first `<Security>` header block is removed) and sent to the final destination. The following shows this message, which is sent from the firewall to the service.

```
(001) <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
(002)   <S:Header>
(003)     <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
(004)       <m:action>http://fabrikam123.com/getQuote</m:action>
(005)       <m:to>http://fabrikam123.com/stocks</m:to>
(006)       <m:rev><m:via/></m:rev>
(007)       <m:from>mailto:johnsmith@fabrikam123.com</m:from>
(008)       <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
(009)     </m:path>
(010)     <wsse:Security
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
            S:mustUnderstand="1">
(011)       <wsse:BinarySecurityToken
              EncodingType="wsse:Base64Binary"
              wsu:Id="X509Token4Stockquote"
              ValueType="wsse:X509v3">
(012)          MIIDQTCCAqqgAwIBAgICAQQwDQYJKoZIhvcNAQEF...
(013)       </wsse:BinarySecurityToken>
(014)       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
(015)         <SignedInfo>
(016)           <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
(017)           <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
(018)           <Reference URI="">
(019)             <Transforms>
(020)               <Transform Algorithm="http://...#RoutingTransform"/>
(021)               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
(022)             </Transforms>
(023)             <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
(024)             <DigestValue>yH8GsCH3FT7qqSTj7CM=</DigestValue>
(025)           </Reference>
(026)         </SignedInfo>
(027)         <SignatureValue>
(028)             9OZ21MKnCQi2Jglw2qIO5e1azezl/j0BP3h2Ut...
(029)         </SignatureValue>
(030)         <KeyInfo>
(031)           <wsse:SecurityTokenReference>
(032)             <wsse:Reference URI="#X509Token4Stockquote"/>
(033)           </wsse:SecurityTokenReference>
(034)         </KeyInfo>
(035)       </Signature>
```

```
(036)     </wsse:Security>
(037)   </S:Header>
(038)   <S:Body>
(039)     <m:GetLastTradePrice xmlns:m="http://www.fabrikam123.com/">
(040)       <symbol>DIS</symbol>
(041)     </m:GetLastTradePrice>
(042)   </S:Body>
(043) </S:Envelope>
```

The firewall optionally adds new security information by prepending a new security token to the `<Security>` header block for the ultimate destination.

Additional scenarios might include the selective routing of messages within a corporate intranet based on the sender. In some cases messages might be sent to a server within the DMZ or protected region within a corporate intranet for various reasons including security and performance.

Contributors
This document was jointly authored by IBM and Microsoft.

Key contributes include (alphabetically): Giovanni Della-Libera, Microsoft; Praerit Garg, Microsoft; Maryann Hondo, IBM; Chris Kaler, Microsoft; Hiroshi Maruyama, IBM; Anthony Nadalin, IBM; Nataraj Nagaratnam, IBM; John Shewchuk, Microsoft; Dan Simon, Microsoft.

Resources

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

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

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

WSSEC
IBM, Microsoft and Verisign, "Web Services Security (WS-Security)," 05 April 2002

WSSECA
IBM, Microsoft and Verisign, "Web Services Security (WS-Security) Addendum," TBD