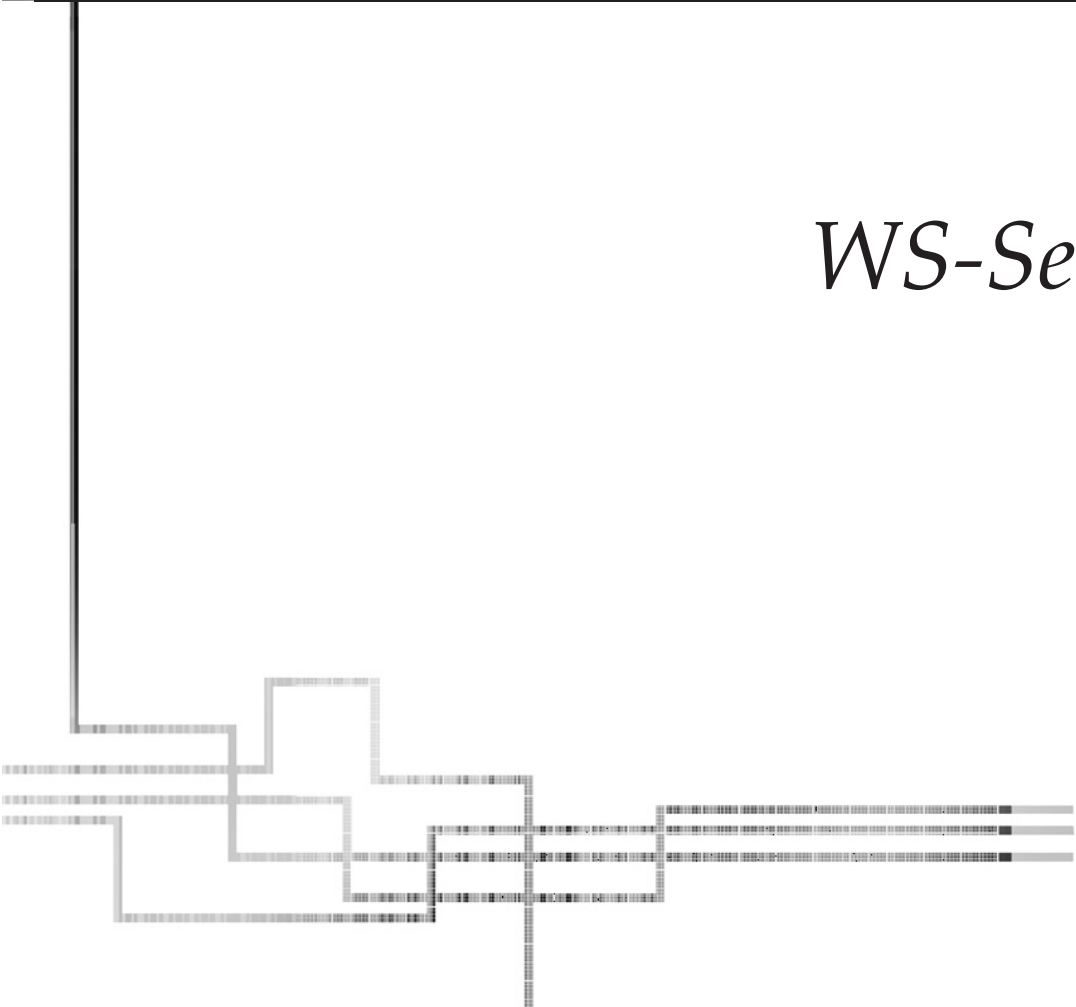




CHAPTER 9

WS-Security



So far, the technologies that we have covered in this book have been either primarily for XML security (for example, XML Signature and XML Encryption) or applicable to the advantages of XML to information security functionality such as key management or access control rules (for example, XKMS, XACML). WS-Security, by contrast, is primarily for securing SOAP messages. It addresses the SOAP security issues discussed in Chapter 3. We've seen that technologies such as XML Signature, XML Encryption, and SAML can be used for purposes other than Web Services security. WS-Security explains how they are used for Web Services security in particular.

WS-Security defines how security tokens are contained in SOAP messages, and how XML Security specifications are used to encrypt and sign these tokens, as well as how to sign and encrypt other parts of a SOAP message. In practice, this means defining the XML elements and attributes that are used to enclose tokens into SOAP messages, and the means to enclose XML Signature and XML Encryption into SOAP.

WS-Security is part of a road map from IBM and Microsoft that includes later specifications such as WS-Trust, WS-Policy, and WS-SecureConversation. It can be used apart from these specifications, but it should be understood in the full context of the "WS-*" specifications.

This chapter uses the Web Services Enhancements 1.0 for Microsoft .NET (WSE) for hands-on familiarity with WS-Security.

INTRODUCTION TO WS-SECURITY

WS-Security has undergone a number of incarnations. It was initially released by Microsoft in October 2001. Then, in April 2002, IBM and Microsoft released their joint "Security in a Web Services World" document. This defined a security framework for Web Services, the first of which to be released (in conjunction with VeriSign) is WS-Security. Later specifications for Web Services security include WS-Trust, WS-Policy, and WS-SecureConversation. June 2002 saw WS-Security submitted to the OASIS standards body, home of SAML and XACML. A Web Services Security group was formed in OASIS in order to develop WS-Security as an OASIS standard.

The definitions of element names for packaging security tokens into SOAP messages are the practical "nuts and bolts" part of WS-Security. Sitting above this is a conceptual model that abstracts different security technologies into "claims" and "tokens." The additional security road map specifications build on these concepts, solidifying them into XML specifications and explaining how to apply for a security token, how tokens are linked to identity, and how security information may be associated with a Web Service.

WS-Security Abstractions

Web Services are designed to allow software from disparate companies to communicate together. They provide a level of abstraction above different platforms and programming languages, allowing different systems to communicate in a loosely coupled fashion. As well as using different platforms and programming languages, communicating companies

may well use different security technologies. One company may use Kerberos, while another may consume X.509 certificates. Just as Web Services themselves provide a level of abstraction for companies to link their business logic, the IBM/Microsoft *Security in a Web Services World* road map provides a level of abstraction for companies using different security technologies to communicate securely using SOAP. This level of abstraction means not only that existing security infrastructure can be used for Web Services security, but that new security technologies can also be incorporated.

Tokens and Claims

Claims are statements about a subject either by the subject or by a relying party that associates the subject with a property, such as identity or entitlements. Read the preceding sentence again, because this can seem like a roundabout definition. A claim says something about a subject (end user or entity) that may be used for an access control decision. A token is an XML representation of security information, including claims. A token may either be signed or unsigned. An example of an unsigned token would be a password or symmetric encryption keys used as shared secrets. Examples of signed security tokens are X.509 digital certificates (which are signed by a certificate authority) or a Kerberos ticket. If an unsigned token is used, then confidentiality must be assured to ensure that the token is not intercepted by a third party. As we learned in Chapter 2, when sending a security token, proof of possession of the token must be ensured. It isn't as simple as putting an X.509 certificate into a SOAP message, because a third party could take that X.509 certificate and copy and paste it into another message.

The WS-Security model also caters to SOAP endpoints and intermediaries. The model defines scenarios where the integrity and confidentiality of SOAP messages is ensured while the messages traverse intermediaries. In addition, it describes scenarios where the intermediaries themselves perform security functionality—for example, a SOAP “firewall.” Additional specifications, including WS-Trust and WS-Policy, will define how security tokens are issued. The request and issuance of security tokens will also make use of Web Services. Let's look at these additional specifications first, before diving into WS-Security itself. It is important to frame WS-Security in the context of the IBM/Microsoft Web Services Security Roadmap.

IBM/Microsoft Web Services Security Road Map

WS-Security is just the first of the Web Services security specifications to be released as part of the IBM and Microsoft Web Services security road map. These specifications are shown in Figure 9-1.

Looking at Figure 9-1, you can see that the specifications are being produced from the bottom up. SOAP is at the base of the diagram. There is nothing under SOAP, of course, because SOAP is transport-independent. WS-Security sits above SOAP in the diagram because it provides a means of encrypting and signing portions of a SOAP message, using XML Signature and XML Encryption, and for enclosing security tokens in a SOAP message to represent claims.

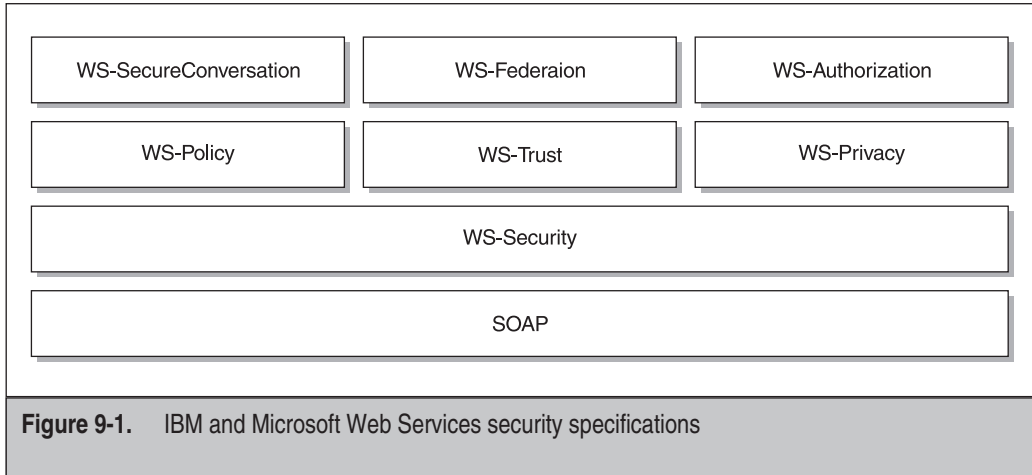


Figure 9-1. IBM and Microsoft Web Services security specifications

Walking through this stack of Web Services security specifications from bottom to top, we can see that each specification depends on its predecessors, mutually building a complete security context for Web Services.

WS-Policy

WS-Policy allows organizations that are exposing Web Services to specify the security requirements of their Web Services. These security requirements include the supported algorithms for encryption and digital signatures, privacy attributes (such as which parameters must be encrypted), and how this information may be bound to a Web Service. The binding to a Web Service will most likely take the form of a WSDL binding, in order to attach policy information to the definition of Web Service. WS-Policy allows organizations initiating a SOAP exchange to discover what type of security tokens are understood at the target, in the same way that WSDL describes a target Web Service. For example, one organization may only consume Kerberos tickets while another organization may only understand X.509 certificates.

WS-Trust

WS-Trust defines how trust relationships are established. Trust relationships can be either direct or brokered. In the case of brokered trust, a “trust proxy” is used to read the WS-Policy information and request the appropriate security token for enclosure in the SOAP message. A trust proxy requests security tokens from an issuer of security tokens. WS-Security will be used to transfer the required security tokens, making use of XML Signature and XML Encryption to ensure their integrity and confidentiality.

This trust model allows *delegation* and *impersonation*. This means that the trust proxy can insert security tokens into the SOAP message that represent the end user. Remember from Chapter 3 that because SOAP is not sent directly by end users, but on their behalf, the SOAP message must contain security information that maps back to the end user.

WS-Privacy

We saw in Chapter 3 that privacy is implemented using confidentiality and access control, and concerns the context and sensitivity of information that is being communicated. Similarly, WS-Privacy uses a combination of WS-Policy, WS-Security, and WS-Trust to communicate privacy policies. These privacy policies are stated by organizations that are deploying Web Services, and require that incoming SOAP requests contain claims that the sender conforms to these privacy policies. The WS-Security specification is used to encapsulate these claims into security tokens, which can be verified.

WS-Privacy explains how privacy requirements can be included inside WS-Policy descriptions. WS-Trust is used to evaluate the privacy claims encapsulated (using WS-Security) within SOAP messages against user preferences and organizational policy. This “Russian doll” model means that the “WS-*” specifications depend on each other.

WS-SecureConversation

SSL is widely used for point-to-point authentication and confidentiality of Web Services communications. We saw in Chapter 3 that message-level security is required for SOAP traffic that may traverse intermediaries. In addition, Web Services traffic cannot be guaranteed to use HTTP, so relying on SSL for authentication and confidentiality is not appropriate. WS-Security defines the use of security tokens within SOAP messages, when combined with XML Signature and XML Encryption, to provide proof of possession and confidentiality of the claims that these tokens encapsulate. When a SOAP message is received, the security token is evaluated and checked against a security policy. However, this process must be repeated for each incoming SOAP message. This is obviously a performance issue, because there is no concept of a session for a group of SOAP messages.

WS-SecureConversation fills this gap, by allowing a requestor and a Web Service to mutually authenticate using SOAP messages and establish a mutually authenticated security context. This security context uses session keys, derived keys, and per-message keys. Like SSL, WS-SecureConversation builds upon the fact that symmetric encryption is faster than asymmetric encryption. By going through the process of using asymmetric encryption to negotiate a symmetric key *once*, this key can be used for a series of SOAP messages. This means that each SOAP message does not have to go through a lengthy and intensive process of message-level authentication. The choreography of the session key negotiation is likely to be similar to that of SSL, meaning that the description of WS-SecureConversation as “SSL at the SOAP level” is valid.

WS-SecureConversation builds upon WS-Security and WS-Trust to securely exchange context (collections of claims about security attributes and related data) in order to negotiate and issue keys. WS-SecureConversation is designed for the SOAP message layer, where messages may traverse a variety of transports (SMTP, HTTP, and so forth) and intermediaries (not all of whom may be trusted). The use of WS-SecureConversation does not preclude the use of transport-level security across point-to-point links.

WS-Federation

Like WS-SecureConversation, WS-Federation also builds upon the specifications that underpin it. It explains how federated trust scenarios may be constructed using WS-Security, WS-Policy, WS-Trust, and WS-SecureConversation. “Federation” in this case involves brokering between different security specifications—for example, communication between a party who understands Kerberos and another party who understands X.509 digital certificates to allow an end user to authenticate to one party, but then use a Web Service exposed by the other party. WS-Policy and WS-Trust are used to determine which tokens are consumed, and how to apply for tokens from a security token issuance service. WS-Federation acts at a layer above WS-Policy and WS-Trust, indicating how trust relationships are managed.

WS-Authorization

This specification has a number of overlaps with XACML, which we encountered in Chapter 7. WS-Authorization describes how access policies for a Web Service are specified and managed. This specification is flexible and extensible with respect to both authorization format and authorization language. It supports both ACL-based authorization and RBAC-based authorization.

WS-Security Elements and Attributes

The WS-Security specification defines XML elements and attributes to enclose security tokens inside SOAP messages, and describes how XML Signature and XML Encryption can be used for confidentiality and integrity of these tokens (and other content) within the SOAP messages. Information is grouped together within blocks. There is a block for a username-and-password combination, a block for a binary security token (for example, an X.509 certificate), and blocks for encrypted and signed information. Let’s walk through these blocks.

Security Block

The Security element in WS-Security is contained within the SOAP header. It is structured within the SOAP message as follows:

```
<S:Envelope>
  <S:Header>
    ...
    <Security S:actor="http://www.vordel.com/appml/" S:mustUnderstand="1">
      ...
    </Security>
    ...
  </S:Header>
  ...
</S:Envelope>
```

In the example, the SOAP mustUnderstand attribute is set to 1 to indicate that the Security header entry is mandatory for the recipient to process. If the recipient cannot process the security information, the processing will fail.

Because a single SOAP message may contain more than one Security header block, targeted at separate receivers, the SOAP actor attribute is used to indicate which security tokens are targeted at which Web Services. This is necessary when a SOAP message is routed through at least one intermediary on the way to its endpoint. The SOAP actor attribute is not mandatory for the Security element. However, if more than one Security header block is present in a SOAP message, then only one Security header block can omit the SOAP actor attribute and no two Security header blocks can have the same SOAP actor value. A single Security block cannot contain security tokens targeted at different recipients. If no SOAP actor attribute is present in the Security element, the Security block can be consumed by any intermediary but may not be removed by any of the intermediaries.

As well as enclosing security tokens, the Security header block presents information about the use of XML Signature and XML Encryption in the SOAP message. An XML Signature or XML Encryption block within the SOAP message may refer to another section of the Security block. This situation occurs, for example, if an XML Signature KeyInfo section, contained within the Security block, references an X.509 certificate that is also contained within the Security block. When a subelement of a Security block refers to another subelement, an ID is used to link the two together. We will see an example of this later in this chapter, in the section entitled “BinarySecurityToken Used with XML Signature.”

UsernameToken

The UsernameToken block defines how username-and-password information is enclosed within SOAP. As we saw in Chapter 3, SOAP messages are not sent from end users, so it will not be usual for passwords typed by end users to find their way into SOAP messages. If end users are authenticating using username and password, then it is more appropriate to issue a SAML authentication assertion or Kerberos ticket in order to represent the user’s authentication act. However, two companies may agree on the use of a username/password combination as a shared secret, to be used for authentication of SOAP messages.

The following listing shows an example of a UsernameToken block within a SOAP message:

```
<S:Envelope xmlns:S=http://www.w3.org/2001/12/soap-envelope
  xmlns:wss="http://schemas.xmlsoap.org/ws/2002/07/secext">
  <S:Header>

    <wss:Security>
<wss:UsernameToken xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
  <wss:Username>NNK</wss:Username>
```

```

    <wsse:Password Type=" wsse:PasswordDigest ">FEdR...</wsse:Password>
    <wsse:Nonce>FKJh...</wsse:Nonce>
    <wsu:Created>2001-10-13T09:00:00Z </wsu:Created>
  </wsse:UsernameToken>
    </wsse:Security>
    ...
  </S:Header>
  ...
</S:Envelope>

```

In this listing, the SOAP envelope is defined first, with the WS-Security block placed in the header of the message. Notice that the namespace is “http://schemas.xmlsoap.org/ws/2002/07/secext.” That is because the WS-Security Addendum, issued by Microsoft, VeriSign, and IBM in August 2002 (and not July, as the namespace suggests) is being used. The August 2002 Addendum builds upon the April 2002 WS-Security specification in order to allow for the use of a nonce (“number once”—see Chapter 2) and timestamps within UsernameToken in order to guard against replay attacks. If these were not used, there would be a danger that the message could be intercepted by a third party, re-sent, and re-authenticated. The use of transport and session layer security also provides point-to-point authentication and confidentiality, and WS-SecureConversation allows for the use of a session that traverses SOAP intermediaries. The password, in this case, is a shared secret between the requestor and the Web Service, which must be protected from eavesdroppers or potential replay attackers.

As we learned in Chapter 2, the use of a nonce or timestamp must be accompanied by a digital signature so that an intruder cannot simply change the value of the nonce or timestamp without detection. In the “BinarySecurityToken Used with XML Signature” section, we will see how an XML Signature is enclosed within a WS-Security block.

BinarySecurityToken

A UsernameToken, as we’ve just seen, encloses XML data. However, not all security data may be as easily enclosed in XML. If a claim is based on a binary token, such as an X.509 digital certificate or a Kerberos ticket, a different encoding is required. Therefore, WS-Security defined a BinarySecurityToken structure to enclose non-XML security tokens. It is formed as follows:

```

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

```

The Id attribute of the security token is used for referencing from elsewhere in the SOAP message. We will see in the next subsections how it is used in the context of XML Signature, where a signature points to an X.509 certificate that may be used to validate the signature. The ValueType attribute contains a qualified name that defines the value type of the encoded binary data. It may make use of XML namespaces. Examples of ValueType include wsse:X509v3 for an X.509 v3 digital certificate, wsse:Kerberosv5TGT for a Kerberos ticket-granting ticket, and wsse:Kerberosv5ST for a Kerberos ticket.

The `EncodingType` attribute is used to specify the encoding format of the binary data (for example, `wsse:Base64Binary` for base64-encoded binary data or `wsse:HexBinary` for XML Schema hex encoding). An example `BinarySecurityToken` is shown here:

```
<wsse:BinarySecurityToken
    xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
    Id="myToken"
    ValueType="wsse:X509v3"
    EncodingType="wsse:Base64Binary">
    MIEZzCCA9CgAwIBAgIQEmtJZc0...
</wsse:BinarySecurityToken>
```

NOTE XML Signature also provides mechanisms for encoding X.509 certificates, using `ds:KeyInfo`, and this may provide additional flexibility.

Let's look now at how the `BinarySecurityToken` is used with XML security within SOAP, beginning with XML Signature.

BinarySecurityToken Used with XML Signature

Binary security tokens include Kerberos tickets and X.509 digital certificates. When combined with proof of possession of the key associated with the security token—for example, using an XML Signature—binary security tokens may be used for authentication.

Let's see an example of a `BinarySecurityToken` block being referenced by an XML Signature block within a SOAP message:

```
<?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/2002/07/secext"
    xmlns:xenc="http://www.w3.org/2001/04/xmenc#">
  <S:Header>

    <wsse:Security>
      <wsse:BinarySecurityToken
        ValueType="wsse:X509v3"
        EncodingType="wsse:Base64Binary"
        Id="X509Token">
        MIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
      </wsse:BinarySecurityToken>
    <ds:Signature>
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm=
          "http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:SignatureMethod Algorithm=
          "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
```

```

    <ds:Reference URI="#bodydata">
      <ds:Transforms>
        <ds:Transform Algorithm=
          "http://www.w3.org/2001/10/xml-exc-c14n#" />
      </ds:Transforms>
      <ds:DigestMethod Algorithm=
        "http://www.w3.org/2000/09/xmldsig#sha1" />
      <ds:DigestValue>EULddytSods:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    BL8jdfToEbl1/vXcMZNNjPOVEWRj3dfj32lsf2weWE
  </ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#X509Token" />
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</S:Header>
<S:Body>
  <tru:StockSymbol Id="bodydata" xmlns:tru="http://quotes.com/payload">
    QQQ
  </tru:StockSymbol>
</S:Body>

```

To understand the preceding listing, work from the bottom up. The “StockSymbol” child of the SOAP Body element contains an Id attribute. The content of this attribute is “bodydata.” Look at the XML Signature data that is contained within the Security block inside the SOAP header. The Reference section of the XML Signature points to the “bodydata” Id attribute of the StockSymbol element in the SOAP body. This is what is being signed. By looking at the XML Signature block in the Security block, you can see that it is familiar from Chapter 4. It contains a digest of the signed data, information about which algorithms are used, and, of course, the signature itself.

The KeyInfo section of the XML Signature points to a WS-Security BinarySecurity Token, using the new SecurityTokenReference element. The URI attribute of the Reference subelement of the SecurityTokenReference points to the X.509 certificate contained at the top of the SOAP message. The X.509 certificate carries the Id “X509Token” to link to the XML Signature.

We have seen that it is often the best practice to discard the X.509 certificate within the SOAP message, and instead pull the X.509 certificate from an LDAP directory. In that case, however, the KeyInfo section provides information about which X.509 certificate to retrieve in order to validate the signature.

BinarySecurityToken Used with XML Encryption

A `BinarySecurityToken` block can also be associated with an XML Encryption block. We saw in Chapter 5 that XML Encryption contains a `KeyInfo` section, which indicates which public key was used to encrypt the symmetric key that was used to encrypt the data. The following code listing is adapted from the WS-Security specification. Notice that the `KeyInfo` contains distinguished name information from an X.509 certificate. This indicates to the recipient that the corresponding private key is to be used to decrypt the symmetric key used to decrypt the data. Notice that the XML Encryption data in the Security block points to the encrypted data within the body of the SOAP message using the "enc1" Id.

An XML Signature is also used in this example. A transform called `RoutingTransform` is intended to isolate the routing information and the body of the SOAP message. This transform isolates the information which is signed.

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <S:Header>
    <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
      <m:action>http://quotes.com/getQuote</m:action>
      <m:to>http://quotes.com/stocks</m:to>
      <m:from>mailto:mark@vordel.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=Mark O'Neill, O=Vordel, C=US</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>
</S:Header>
<S:Body>
  <xenc:EncryptedData
    Type="http://www.w3.org/2001/04/xmlenc#Element"
    Id="enc1">
    <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc" />
    <xenc:CipherData>
```

```
<xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
</S:Body>
</S:Envelope>
```

Error Handling in WS-Security

There are a number of errors that can occur when a SOAP message formatted using WS-Security is processed. These include the following scenarios:

- **Security token type unsupported** Note that the use of WS-Policy and WS-Trust will allow organizations to communicate information about which types of security tokens they can understand.
- **Invalid security token** This error occurs, for example, if a security token has been corrupted en route to the recipient. It also occurs if the signature over the security token does not validate, or if an encrypted security token cannot be decrypted.
- **Security token cannot be authenticated** If an X.509 certificate is received in a BinarySecurityToken block and the issuer cannot be determined, or if the certificate does not match those that are contained in a local LDAP directory for authentication, then this is the appropriate error type.
- **Referenced security token unavailable** This would occur, for example, if an XML Signature KeyInfo section references an X.509 digital certificate, but this certificate is not present in the SOAP message.

Notice that some of these errors involve failures of processing, whereas others involve unsupported security tokens. SOAP Faults, as defined in the SOAP 1.1 and SOAP 1.2 specifications, are used by a recipient to indicate to the sender that an error occurred processing a message.

SOAP Fault includes a FaultCode parameter that conveys the type of error that occurred. When an unsupported token is received, the `wsse:UnsupportedSecurityToken` fault code should be used. If an unsupported cryptographic algorithm is referenced in the SOAP message, the fault code should be `wsse:UnsupportedAlgorithm`.

TIP If a message cannot be processed due to a failure of signature or decryption, there is a chance that the incoming message may have been a denial of service attack or a cryptographic “clogging” attack designed to cause disruption to a signature validation or decryption application. In that case, it would be unwise to provide reinforcement to the sender, so it may be wiser to simply not return an error message to the sender of the SOAP message.

If a SOAP Fault message is to be returned due to a processing failure, the fault codes to use are as follows:

- **wsse:InvalidSecurity** This fault code may be used if the contents of the Security block in the SOAP header cannot be processed.
- **wsse:InvalidSecurityToken** This fault code is self-explanatory—the security token provided in the SOAP message may be invalid due to a broken signature, or due to corruption of the token itself.
- **wsse:FailedAuthentication** This is used if the security token cannot be authenticated or authorized.
- **wsse:SecurityTokenUnavailable** This is used if the security token referenced in the SOAP message is not available in the SOAP message or from another location, such as an LDAP directory or an XKMS service.

SAML AND WS-SECURITY

During the early part of 2002, when SAML and WS-Security were receiving a lot of press attention, it was commonplace for journalists to write articles with a “WS-Security vs. SAML” theme. This was misleading because WS-Security and SAML solve different problems: SAML explains how security assertions may be expressed in XML format, whereas WS-Security explains how security information is contained in SOAP messages.

The WS-Security Profile for XML-based Tokens, published in August 2002, explains how SAML information is enclosed inside SOAP messages. A “SOAP binding” for SAML was lacking in the SAML 1.0 specification, so this is now provided by WS-Security.

The following code listing explains how a SAML v1.0 assertion is contained within a SOAP message:

```
<?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">
  <S:Header>
    <wsse:Security>
      <saml:Assertion
        MajorVersion="1"
        MinorVersion="0"
        AssertionID="SecurityToken-ef375268"
        Issuer="CompanyX"
        IssueInstant="2002-07-23T11:32:05.6228146-07:00"
        xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
        ...
      </saml:Assertion>
      ...
    </wsse:Security>
  </S:Header>
</S:Envelope>
```

```
        </wsse:Security>
    </S:Header>
    <S:Body>
    </S:Body>
</S:Envelope>
```

The SAML assertion is contained within a Security block, which is contained inside the SOAP header. A SAML assertion may be digitally signed or encrypted, in a similar manner to how other security tokens may be signed and encrypted using WS-Security. The processing of a SAML assertion, contained in a WS-Security formatted SOAP message, should not be different from the processing of any other type of security token expressed using WS-Security.

Code Example: Using the Microsoft WSE

The Microsoft WSE is a downloadable tool that allows Visual Studio .NET developers to build Web Services applications that make use of technologies such as WS-Security. It is available at the following URL: <http://msdn.microsoft.com/webservices/building/wse>.

The code examples provided with the Microsoft WSE make use of C# as their programming language. The .NET Common Language Framework (CLR) is used. Let's look at a code example for the creation of a SOAP message containing a digital signature and an X.509 digital certificate.

First, the security and XML processing functionality from the .NET platform is required for this program, so the following code pulls in the required packages:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Security.Cryptography;
using System.Text;

using Microsoft.Web.Services;
using Microsoft.Web.Services.Security;
using Microsoft.Web.Services.Security.X509;

using Microsoft.Web.Services.QuickStart.X509;
```

Let's look at code that calls a Web Service that adds two numbers, similar to the Web Service we encountered in Chapter 3. This code is based on an example provided with the WSE in the `\Samples\QuickStart\Clients\X509Signing\` folder.

```
// Instantiate an instance of the web service proxy
AddNumbers serviceProxy = new AddNumbers();
SoapContext requestContext = serviceProxy.RequestSoapContext;
// Configure the proxy
```

```
ConfigureProxy(serviceProxy);
// Get our security token
X509SecurityToken token = GetSecurityToken();
if (token == null)
    throw new ApplicationException("No key provided for signature.");
// Add the signature element to a security section on the request
// to sign the request
requestContext.Security.Tokens.Add(token);
requestContext.Security.Elements.Add(new Signature(token));
// Call the service
Console.WriteLine("Calling {0}", serviceProxy.Url);
int sum = serviceProxy.AddInt(a, b);
```

You can see that the WS-Security model is reflected in the preceding code listing. An X.509 security token is requested and then a digital signature is added to the SOAP message, referencing this X.509 token. The code to request the X.509 certificate is shown here:

```
public X509SecurityToken GetSecurityToken()
{
    X509SecurityToken securityToken;
    // open the current user's certificate store
    // X509CertificateStore store =
        X509CertificateStore.CurrentUserStore(X509CertificateStore.MyStore);
    bool open = store.OpenRead();
    try
    {
        // Open a dialog to allow user to select the certificate to use
        //
        StoreDialog dialog = new StoreDialog(store);
        Microsoft.Web.Services.Security.X509.X509Certificate cert = null;
        cert = dialog.SelectCertificate(IntPtr.Zero, "Select
Certificate", "Choose a Certificate below for signing.");
        if (cert == null)
        {
            throw new ApplicationException("You chose not to select an X509
certificate for signing your messages.");
        }
        else if (!cert.SupportsDigitalSignature || !cert.key == null)
        {
            throw new ApplicationException("The certificate must support digital
signatures and have a private key available.");
        }
        else
        {
            securityToken = new X509SecurityToken(cert);
        }
    }
}
```

```
finally
{
    if (store != null) { store.Close(); }
}
return securityToken;
}
}
```

A dialog box is used to ask the user to choose which certificate they will employ from their local certificate store. The `StoreDialog` object is used for this purpose. An `X509Certificate` object is used to load the certificate, and it is checked to ensure that it supports the use of a digital signature, and that the corresponding private key is available. As we know from Chapter 2, it is the private key that is used to produce a digital signature, and the corresponding public key may be enclosed with the signature in the message. Providing the private key is available, the X.509 certificate is loaded into an `X509SecurityToken` object and returned.

As you can see from the code listings, the use of WS-Security does not presuppose any knowledge of the structure of the SOAP messages that are created. However, it is important to understand the model and abstractions of WS-Security—the use of tokens and the signing and encryption of these tokens, as well as other data in the SOAP message. The .NET platform allows the developer to take advantage of the power of these abstractions without the requirement to delve into the XML itself.

CHECKLIST

- Read the “Security in a Web Services World” road map document at <http://www-106.ibm.com/developerworks/webservices/library/ws-secmap>. The WS-Security specification is just the first of the specifications in this road map. However, the concepts, including tokens and claims, will carry through into later specifications, such as WS-Policy and WS-Trust.
- Ensure that any Web Services security product your company uses supports WS-Security. WS-Security is arguably the most important Web Services security specification, because it explains how XML security relates to SOAP, and will underpin many later specifications.