Chapter 8

Web Services Security

Web services provide significant new benefits for SOA-based applications, but they also expose significant new security risks. Creating and managing a secure Web services environment involves dealing with various Internet, XML, and Web services security mechanisms. Other security mechanisms may be already in place within the execution environment, especially when existing systems become service-enabled to join the SOA.

The general approach is relatively straightforward, taking into account:

- Transport-level security such as firewalls, virtual private networks, basic authentication, non-repudiation, and encryption.
- Message-level security such as using authentication tokens to validate requester identity and authorization assertions to control access to provider services.
- Data-level security such as encryption and digital signature to protect against altering stored and/or transmitted data.
- Environment-level security such as management, logging, and auditing to identify problems that need to be fixed and establishing trusted relationships and communication patterns.

Achieving the right mixture of the various technologies and levels of protection, and figuring out what threats to protect against and how, typically takes some time and effort. A good solution protects programs and data against unauthorized access, guards against the possible consequences of in-flight message interception, and prevents a variety of malicious attacks that have become all too familiar in the Internet world.

This chapter:

- Describes the various threats and challenges that need to be guarded against.
- Summarizes the basic Web services security technologies.
- Provides detail on some of the more important technologies and standards.

Most of the technologies described in this chapter were designed and developed specifically for use with Web services. However, several of them are generic security mechanisms that can be applied to Web services. As a rule, the WS-Security framework describes how to incorporate these other security technologies into Web services by defining a place for them within SOAP headers.

Figure 8-1 illustrates the fact that WS-Security provides a framework into which other security technologies are plugged. For example, WS-Security does not define any authentication ticket mechanism; instead, it defines how to use plain user name/password, Kerberos, and X.509 tickets within the context of a SOAP header. WS-Security also defines how to use XML Signature, XML Encryption, and SAML within SOAP headers.

Other Web services security specifications, such as WS-Trust, WS-Secure-Conversation, and WS-Federation, define protocols that help establish agreements between requesters and providers about the kinds of security they will use. Finally, WS-SecurityPolicy is used to declare a provider’s requirements for security support, such as strong authentication.

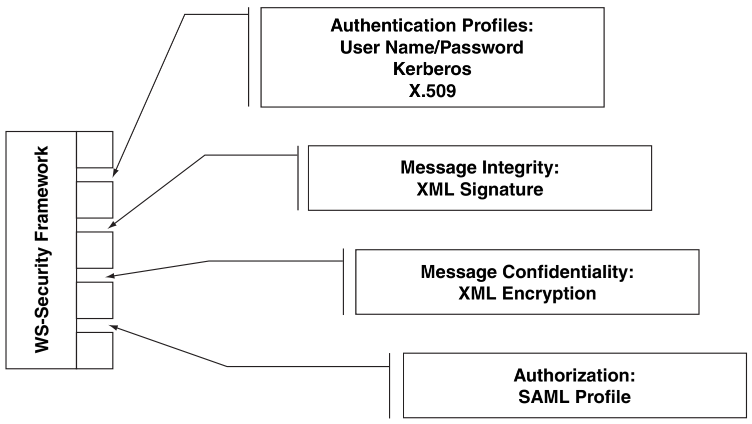

Figure 8-1 Relationship of WS-Security framework to other specifications.

Overarching Concern

Security is sometimes called an "overarching concern" because everything involved in the Web services environment needs some level of protection against the many threats and challenges that IT departments must deal with on a regular basis.

For example, SOAP messages need to be secure, WSDL files may need to be secured against unauthorized access, firewall ports may need additional mechanisms to guard against heavy loads and to inspect Web services messages, and so on. Because Web services are designed for interoperability, an important goal of the security technologies is to enable execution environment technologies to continue to work while adding security mechanisms to the Web services layers above them.

An XML appliance may also be deployed to inspect messages arriving at the edge of the network (that is, where it meets the Internet); if so, this device must be deployed with an understanding or assessment of its relationship to other security mechanisms.

The starting point is ensuring network layer protection using IP Security (IPsec), Secure Sockets Layer (SSL), and basic authentication services, which provide a basic level of protection.

At the next level, WS-Security provides the framework for protecting the message using multiple security technologies. Most of the technologies are defined outside of the WS-Security specification; in that case, WS-Security tells you how to use them within the Web services environment.

Core Concepts

Two basic mechanisms are used to guard against security risks: signing and encrypting messages for data integrity and confidentiality, and checking associated ticket and token information for authentication and authorization. These mechanisms are often used in combination because a broad variety of risks must be taken into account.

As illustrated in Figure 8-2, WS-Security headers can be added to SOAP messages before they are sent to the service provider. The headers can include authentication, authorization,[1] encryption, and signature so that the provider can validate the credentials of the requester before executing the service. Invalid credentials typically result in the return of an error message to the requester. The requester typically adds the authentication and authorization information in the form of tokens. Thus, there's a need to share and coordinate security information, such as tokens, between requester and provider or across a chain of requesters, providers, and possibly SOAP intermediaries.

To successfully manage encryption and authentication for end-to-end message exchange patterns, the WS-Security specification defines several SOAP header extensions. For example:

```
<wsse:Security
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext">
 <wsse:UsernameToken>
  <wsse:Username>Ericn</wsse:Username>
  <wsse:Password>8Bcnu6</wsse:Password>
 </wsse:UsernameToken>
</wsse:Security>
```

[1] Note that as of the time of writing, WS-Authorization was not yet completed.

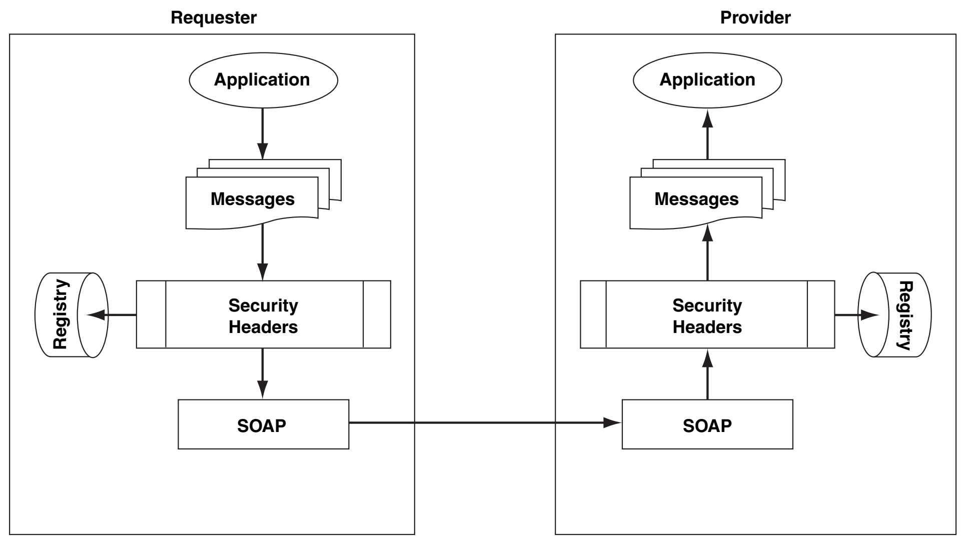

Figure 8-2 Security headers are added to SOAP messages.

The example shows the WS-Security namespace `wsse` and the use of the clear text user name/password authentication feature. The inclusion of WS-Security headers in a SOAP message ensures that the user name/password shown in this example is available for processing by intermediaries as well as at the ultimate destination of the message. Further information on these topics is provided later in this chapter.

If the service provider requires a Kerberos token, the WS-SecurityPolicy declaration associated with the provider's WSDL might look like this:

```
<SecurityToken wsp:Requirement=Kerberos
  <TokenType>... </TokenType>
  <TokenIssuer> ... </TokenIssuer>
</SecurityToken>
```

As shown in Figure 8-3, a key pair (or other encryption mechanism) can be used to encrypt a message before transmission and decrypt it after it's received but before it's processed by the application. Encryption means sending information in code, much like the military does to protect confidential information during wartime. To snoop on an encrypted transmission, someone would have

Newcomer_08.qxd 11/22/04 8:34 AM Page 318

to be able to break the code. Encryption is often used to protect authentication and authorization data (such as a password), even if the data in the SOAP body isn't encrypted.

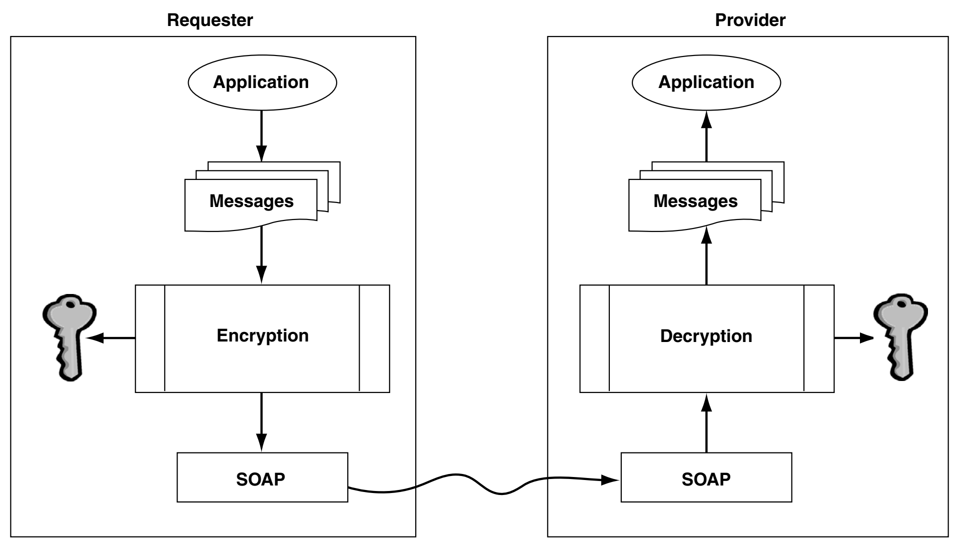

Figure 8-3 Encryption protects messages from snooping.

The encryption information can be sent in a WS-Security header so that a provider knows what encryption algorithm was used to encrypt the message. As with authentication, several standards exist for encryption. Some of the common ones include Secure Shell (SSH) and RSA, named for its inventors, Ron Rivest, Adi Shamir, and Leonard Adelman. RSA developed and first implemented the concepts behind public key cryptography (also called PCKS), which allows services to communicate securely by using a private key and by the exchange of a public key. The private key, which isn't shared, is used in the encryption algorithm, while the public key, which can be shared, is used to decrypt the message. In Web services specifications, the XML Key Management Specification (XKMS) can be used to manage the distribution of public and private keys to enable this style of secure communication.

These basic types of security technologies are also often used in combination with other extended technologies, such as reliable messaging and transactions, to improve the security of an overall system. Transaction processing technologies require additional messages for coordinating transaction protocols, and these often require security to prevent their disruption.

Web services management is often very concerned with security. In addition to securing the management infrastructure itself from unauthorized use (i.e., to prevent anyone from gaining administrative control over the Web services deployment), it's important to be able to monitor and manage the security infrastructure. Web services management tools typically implement some level of security functionality using SOAP intermediaries or SOAP interceptors.

Identity

Identity management for Web services is similar to identity management for any IT system in that the subject (whether a person, machine, program, or abstraction such as a process flow) is given a unique or unambiguous name within the security domain whose validity can be checked. The identity of a Web service requester is sometimes critical for a provider to establish trust because whether or not the requester is allowed to access the provider's service (or any other service, data resource, or device managed by the provider's service) depends upon the identity of the requester.

Identity management is complex for Web services, just like it is for the Web, because Web services can span departments and enterprises. Typically, identity management is performed locally, departmentally, or within an enterprise by ensuring that each employee's user name is unique on the network. Employees are responsible for keeping their passwords private because passwords are used to authenticate the user's identity and to determine the applications, directories, and data the user is allowed to access.

Identity management may need to be performed within a broader scope, such as the Microsoft Active Directory or a corporate-wide LDAP solution. When an identity has to be uniquely managed across the Internet and across enterprises, the level of administration difficulty increases, as does the need for trust. Various initiatives, such as those sponsored by the Liberty Alliance, are focused on establishing mechanisms for identity management for the Internet.

Authentication

Authentication is the process through which an authority verifies a subject's identity, based on some set of proof such as a password or personal identification number (PIN). The authentication process creates a *principal*, which is an object that represents the authenticated subject, such as a credential or token that the subject can use later. On the Web, the subject is typically a user, but for Web services, it can be a machine, program, or other abstract entity represented by the Web service requester. Web services typically use some form of the user name/password mechanism for basic authentication, but stronger forms such as signatures also may be used.

Authentication can be described as the process of confirming that you (or your proxy service requester) are who you say you are. On the Web, this is most often seen as a popup user name/password box, which is called forms-based authentication, which uses a cookie returned on subsequent invocations.[2] Only you know the correct user name and password, so you are authenticating yourself as someone who is allowed to access the Web site. The Web site will have to set up and manage a directory of authorized user name/password combinations so that it can verify the information you submit.

Web services requesters can include authentication information using user name/password information in SOAP headers that the service provider can check against its directory of authorized user name/password combinations. The user ID and password can also be sent via HTTP (no SOAP header required). The provider typically carries out a further refinement of this model to support specific checks for authorization to access specific services or specific data resources. Sometimes requesters are assigned certain roles that can be used as indexes into authorization information—meaning authorization is sometimes carried out according to specific roles such as administrator, clerk, or manager, but again, this is typically managed by the provider and may not appear in the SOAP header (and certainly not in the WS-Security header if it appears at all).

[2] Cookies are not supported in Web services because they are not in XML format and cannot be used across multiple service executions.

Authentication is needed in Web services to verify the identities of the service provider and service requestor. In some cases, mutual authentication may be needed (that is, the provider must authenticate the requester and vice versa).

Digital Signature

A digital signature signs a message digest using a public/private key pair. A hash algorithm creates the message digest, and the encryption algorithm signs the digest (with the private key). The receiver decrypts the signature using the public key, recomputes the message digest, and compares the two. If the message has been altered, the results won't match, and the provider knows the message has been tampered with. As in other encryptions, symmetric or asymmetric key algorithms can be used to compute the signature, although for signing the user of asymmetric keys is more typical.

Summary of Challenges, Threats, and Remedies

This section summarizes the major challenges and threats that need to be addressed using Web services and other security mechanisms and identifies (where possible) the technologies necessary to guard against each challenge or threat.

Web services, because they represent an abstract interfacing and messaging layer, cannot and should not include some of the security mechanisms available within the underlying platforms on which Web services execute. It would be a mistake to try to replicate into the Web services environment such operating system-level protections as memory protection, file or device protection, or even network-level protection.

In general, to guard against the broad variety of threats and challenges, security solutions must be implemented through the transport layer, the Web services layer, and the data layer, and also must be mapped into and out of the underlying execution environment to ensure either that the defined security policy is enforced or that when it is not, there is an audit log entry of the failure or policy breach.

Understanding the Security Architecture

It's important to view the Web services security challenges and threats within their overall architectural context and determine solutions based not simply on a given technology but rather on looking at the overall solution context. That is, you can't just say "use SSL" without understanding the threat you're trying to defend against and without understanding the overall security context into which you'd like to deploy SSL. SSL may be sufficient, but it may not. Multiple security technologies often must be used in conjunction to provide a comprehensive solution to the big security concerns, and it is therefore important to understand how the technologies work together.

The following sections detail some of the specific challenges and threats that the overall Web services security environment must address.

Message Interception

The potential for SOAP message interception and decoding gives rise to a category of security threats that must be guarded against when deploying Web services, including message replay, alternation, and spoofing.

Unless specifically encrypted, Web services messages are transmitted in plain text, which can easily be intercepted and read. An intercepted message can be modified, potentially affecting all or part of the message body or headers. Additional bogus information could be inserted into a message header or body parts. Any message attachment could also be modified or replaced. Altering the message or the attachment could cause bogus information to be sent to and received from a Web service, possibly including a virus. Reading an intercepted message can also give anyone access to confidential information within a message or message attachment, such credit card information, social security numbers, bank account numbers, and so on.

Protecting against message interception includes the use of encryption and digital signatures to preserve confidentiality and integrity.

Person in the Middle Attacks

Because SOAP messages can be routed through intermediaries, and because intermediaries are able to inspect the messages to add or process headers, it's possible for a SOAP intermediary to be compromised. Messages between the requestor and the ultimate receiver could therefore be intercepted while the original parties still believe they are communicating with each other.

Mutual authentication techniques can protect against this type of threat, but signed keys or derived keys provide even better protection.

Spoofing

Spoofing is a complex challenge in which an attacker assumes the identity of one or more trusted (i.e., authenticated) parties in a communication in order to bypass the security system. The target of the attack believes it is carrying on a conversation with a trusted entity. Usually, spoofing is a technique to launch other forms of attack such as forged messages that request confidential information or place fraudulent orders.

It's possible for spoofed Web service messages to include SQL or script tampering to attack through JSP or ASP script execution.

Mutual authentication techniques can protect against this type of threat.

Replay Attacks

A replay attack is one in which someone intercepts a message and then replays it back to the receiver. Replays could also be used to gather confidential information or to invoke fraudulent transactions.

Strong authentication techniques together with message time stamp and sequence numbering can protect against this type of threat.

Denial-of-Service Attacks

When an unauthorized intermediary or other attacker intercepts a SOAP message, the attacker can resend it repeatedly in order to overload the Web services execution environment and effectively deny service to legitimate services that are trying to get through. An attacker can also blast a ton of messages to a Web service after the attacker gets its address. Even if the messages are rejected, the site can get overloaded with error processing.

In general, if someone wants to launch this type of attack, there's no real defense. However, firewall appliances are growing in popularity because they can help mitigate denial-of-service attacks.

Finding the Right Balance

For Web services, as with any application, it's necessary to establish the proper balance between business requirements, protection, performance, and ease of administration. Security mechanisms each carry a performance price not only in terms additional processing overhead when executing a service but also to the IT staff who must design, develop, and administer them. Encryption includes an obvious overhead because it takes time to encrypt and decrypt messages. Sending a user name/password on each message also adds to the overhead of processing a message. It's for these reasons that many of the various technologies have been developed, but this also means that the user needs to understand not only what each technology provides, but also what it costs, and whether or not there are other security mechanisms that can be used and that will satisfy the business requirements while imposing less of a burden.

Securing the Communications Layer

The first level that needs to be secured is the communications transport. In the case of Web services, this is almost always TCP/IP, and this is certainly the case when using HTTP.

Firewalls map a publicly known IP address to another IP address on the internal network, thereby establishing a managed tunnel and preventing access by programs at unauthorized addresses. Web services can work through existing firewall configurations, but this often means increased protection has to be added to firewalls to monitor incoming SOAP traffic and log any problems. Another popular solution involves the use of XML firewalls and gateways that are capable of recognizing Web services formats and performing initial security checks, possibly deployed as intermediaries or within a "demilitarized zone" (i.e., between firewalls).

IP Layer Security

Security mechanisms for the Internet include the IP layer with IP security (IPsec). IPsec provides packet-level authentication and encryption and is typically implemented at the operating system level. IPsec is a facility available to all applications using the Internet, including Web services. However, in practice this means that the IPsec connection is typically part of a separate security setup between the communicating parties. In other words, for Web services to use IPsec, the IPsec communication session has to be established in advance of invoking a Web service, typically by the transport or the user, because nothing in the Web services layer is used to establish an IPsec session.

IPsec is most often used in virtual private network (VPN) applications and between firewalls, which many companies use to secure the communications between remote users and corporate systems. Other VPN technologies are also widely used as a security foundation for Web services invocations, just as they can be used for any other Internet-based application.

Transport-Level Security

At the next level is transport layer security. Typically, this is provided by Secure Sockets Layer/Transport Layer Security (SSL/TLS, usually referred to simply as SSL), which is often seen on the Web as HTTPS. This security level can be implemented in the network application, rather than in the operating system, and Web services can easily and directly use it by requiring HTTPS as a transport. Implementations of SSL for other transports, such as IIOP and RMI/IIOP, also provide the capability for Web services to take advantage of this important privacy mechanism when used over other transports.

SSL provides encryption and strong authentication and may be sufficient for many applications. SSL authentication can be used to provide strong, mutual authentication—much stronger than HTTP Basic, HTTP Digest, or WS-Security user name token authentication. However, SSL is transport-based rather than application-based, so it secures the network nodes rather than the service requester or provider. SSL provides authentication, message confidentiality, and message integrity, but these capabilities are limited to the transport level and cannot be applied to the application level. SSL also does not offer any protection for XML data in storage. It also does not directly support any of the advanced authorization checking such as role-based authorization that many applications may require, although it is possible to map the SSL strong authentication information to a local principal and use that in an application-defined role-based authorization scheme to determine access privileges. But these are application-specific scenarios, not general solutions.

The simplest starting point for Web services security typically is the user name/password checking associated with HTTP that is common to many Web sites. However, basic authentication may not be sufficient for Web services. A password can be encoded using Base64 or another simple obfuscation algorithm even without SSL, but obfuscation does not provide true encryption and therefore is not very secure. When a potential attacker figures out the encoding mechanism or algorithm used for the obfuscation, the message can be intercepted and tampered with. Additional, stronger authentication mechanisms include:

- At the transport level: HTTP Basic, HTTP Digest, and SSL.
- At the application level: User name/password, X.509, Kerberos, SPKM, SAML, and XrML.

What makes an authentication mechanism stronger is mainly its resistance to interception. An authentication token is harder to intercept when it's encrypted or encoded, or when (as with Kerberos and X.509) it's issued by a third party, such as an authentication authority, with whom the application can check to be sure the token is correct.

SSL cannot handle composite applications, however, or complex MEPs. Furthermore, SSL encryption is all or nothing, unlike XML Encryption, which can be applied to any part of an XML document.

Message-Level Security

The next level of security above the transport level is message-level security, where the security protections are provided by the WS-Security framework and associated specifications. The WS-Security framework defines SOAP headers that include the necessary information to protect messages. The WS-Security specification defines the security header for SOAP messages and what can be included within the header. Associated specifications define the contents of the SOAP security header and the processing rules associated with those contents.

Because Web services expose access to programs and data stores, their use creates additional requirements for security protection. Furthermore, complex Web services may span multiple network locations discovered dynamically or composed into a larger interaction such as a process flow. Web services need an end-to-end security model for the entire conversation because sensitive information could be passed from service to service. A Web service interaction also may potentially involve multiple parties using different security-related technologies.

The WS-Security Framework

WS-Security (Web Services Security) is the name of a set of specifications[3] that augment SOAP message headers to incorporate solutions to many common security threats, in particular those related to the requirements of Web services messaging.

Because SOAP is a particular form of an XML document designed for messaging and interoperability, WS-Security needs to define how XML Encryption and XML Signature should be used with SOAP—this is one of the major motivations for WS-Security.

[3] See http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

The WS-Security specifications protect against:

- **Message alteration**—By including digital signatures for all or parts of the SOAP body and the SOAP header.
- **Message disclosure**—By supporting message encryption.

The WS-Security framework can also be used to:

- Preserve message integrity through the use of strong key algorithms.
- Authenticate messages through the use of various token mechanisms such as Kerberos and X.509.

The specifications are divided between the core framework and profiles of other technologies that are defined to fit within the framework. Much of the work of the WS-Security Technical Committee at OASIS is involved in adding profiles to the WS-Security suite of specifications.

At the time of writing, the WS-Security framework consists of the following specifications:

- Web Services Security: SOAP Message Security.
- Web Services Security: User name Token Profile.
- Web Services Security: X.509 Token Profile.
- Profiles for the use of other technologies with the framework are under development, including SAML, Kerberos, and XrML (Extensible Rights Markup Language for role-based authorization and control of access to digital media and services content).

The WS-Security set of specifications defines the overall framework for including these various types of security information into SOAP headers and then defines (or profiles) other technologies so that both requesters and providers can share a common understanding of their use.

The WS-Security framework is designed to work with SOAP 1.1 and SOAP 1.2[4] by defining the security tokens and encryption mechanisms that go in the SOAP headers.

WS-Security 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., to support multiple security token formats).

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<s11:Envelope
 xmlns:s11="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://www.docs.oasis-open.org/wss/2004/01/
             oasis-200401-wss-wssecurity-secext-1.0.xsd">
 <s11:Header>
  <wsse:Security>
      ...
  </wsse:Security>
 </s11:Header>
 <s11:Body>
  ...
 </s11:Body>
</s11:Envelope>
```

This example illustrates a SOAP 1.1 message (indicated by the `S11`: namespace prefix) and shows the OASIS standard namespace for the WS-Security security header (indicated by the `wsse`: namespace prefix). The namespace for SOAP 1.2 is as follows:

```
xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
```

WS-Security describes a mechanism for encoding binary security tokens within the header. The specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also

[4] Note that SOAP with Attachments is supported for SOAP 1.1 only because SOAP 1.2 includes its own binary message attachment format (MTOM—message transmission optimization mechanism).

includes extensibility mechanisms that can be used to further describe the characteristics of the security tokens that are included with a message.

A security token is a credential that proves identity. When using Kerberos, X.509, SAML, or XrML, you've already proven your identity at least once. The question is whether the service will accept this credential (typically based on the trust relationship that exists between the issuing authority and the service and on the freshness of the credential). Sometimes the service requires additional proof (such as a signature) or multiple forms of proof. WS-Trust provides a protocol for the service to request additional proof. Assuming that the service accepts the credentials, it then determines whether the subject has permission to access the requested resource.

There are many different ways of managing these tokens on a network, and there are also different ways of proving an identity. With a user name, for example, an accompanying password proves that this is really your identity. A Kerberos ticket, however, is encrypted by its issuer using a key that the service provider can verify. Additionally, a digital signature might be sent along with a certificate to authenticate an identity.

WS-Security therefore needs to support multiple approaches for conveying and authenticating identity. WS-Security doesn't define how to perform authentication but rather defines how to transmit a variety of different security tokens within the security header. The service provider typically uses the information in the header to authenticate the identity of the requester, but authentication tokens can also be used in other ways.

Although it allows any type of security token, the WS-Security specification explicitly defines four options (user name, binary, XML, and token reference). The simplest (although not always the most secure) option is to send a security token containing a user name and password. To allow this, WS-Security defines a `<UsernameToken>` element that can contain a user name and password.

Because sending unencrypted passwords across a network isn't a very effective authentication mechanism, the `UsernameToken` element is most likely to be used to authenticate SOAP messages sent across an encrypted connection, such as one that uses SSL.

A second option is to send a binary security token, such as the one containing a Kerberos ticket. For example:

```
<wsse:Security
 xmlns:wsse="http://www.docs.oasis-open.org/wss/2004/01/
             oasis-200401-wss-wssecurity-secext-1.0.xsd">
 <wsse:BinarySecurityToken
       ValueType="http://www.docs.oasis-open.org/wss/2004/07/
       ➥oasis-
        000000-wss-kerberos-token-profile-
        1.0#Kerberosv5_AP_REQ"
  EncodingType="http://www.docs.oasis-open.org/wss/2004/01/
  ➥oasis-200401-wss-wssecurity-secext-1.0.xsd#Base64Binary">
   QMwcAG ...
 </wsse:BinarySecurityToken>
</wsse:Security>
```

As this example shows, a Kerberos ticket is sent using the `<BinarySecurity-Token>` element. This element's `ValueType` attribute indicates that this is a Kerberos Version 5 service ticket, which is used to authenticate a client to a particular service. The ticket is encoded using `base64`. Other encoding options have been proposed but not yet defined.

The fourth option is to send a reference to a security token rather than the token itself. WS-Security defines the `<SecurityTokenReference>` element that contains a URI for a security token. The service provider can dereference the URI to obtain the token from its location on the Internet.

Message integrity is provided by leveraging XML Signature in conjunction with security tokens (which may contain or imply key data) to ensure that messages are transmitted without modification. The integrity mechanisms support multiple signatures, including any that might be added by SOAP intermediaries, and are extensible to support additional signature formats.

A signature provides additional proof that the token is valid. The signature verifies the claim that a particular security token represents the producer of the message by using the token to sign a message (demonstrating knowledge of the key). The `<ds:KeyInfo>` element in the signature provides information to the provider as to which key was used to create a signature. The following example illustrates that the sender created the signature using the referenced token (usually a binary token):

```
<ds:KeyInfo>
       <wsse:SecurityTokenReference>
            <wsse:Reference URI="#MyID"/>
       </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

XML Encryption provides message confidentiality in conjunction with security tokens to keep portions of SOAP messages confidential. The encryption mechanisms support additional encryption technologies, processes, and operations by multiple actors. The encryption information references a security token when that token is used to encrypt the data.

WS-Security can handle multiple options for carrying authentication information and ensuring message integrity and confidentiality, but the requester and provider still need a way to agree on which option or options are being used. That's where WS-SecurityPolicy comes in.

WS-SecurityPolicy

Like the WS-Security framework, WS-SecurityPolicy is intended to work with both SOAP 1.1 and 1.2. WS-SecurityPolicy is a policy assertion language that can be used within the WS-PolicyFramework (see Chapter 7, "Metadata Management"). WS-SecurityPolicy can be used to develop an XML-based assertion associated with a Web service endpoint or WSDL file using WS-Policy-Attachment (again, see Chapter 7). A service requester can discover the WS-SecurityPolicy association using the WS-MetadataExchange protocol or can otherwise dereference a URL pointing to the XML file in which the WS-SecurityPolicy assertion is stored.

Decoding the WS-SecurityPolicy assertion tells the service requester what security features are required by the service provider, such as the authentication token format and whether or not the message has to be signed using an XML signature. For example:

```
  <wsp:Policy
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
     xmlns:wssp="http://schemas.xmlsoap.org/ws/2002/12/secext">
```

```
      <wsp:All>
         <wssp:Integrity wsp:Usage="wsp:Required">
         <wssp:Algorithm Type="wssp:AlgSignature"
            URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      </wssp:Integrity>
     <wssp:SecurityToken>
        <wssp:TokenType>wsse:X509v3</wssp:TokenType>
     </wssp:SecurityToken>
   </wsp:All>
 </wsp:Policy>
```

This example shows that the provider requires message integrity using the referenced XML Signature algorithm and requires authentication using X.509 tokens.

By specifying the token type within the `Integrity` element, you are saying that the signature must be created using this type of token. Although not included in this example, the SOAP body contains the Request Security Token (RST) and the Request Security Token Response messages.

WS-Trust

Sometimes it's necessary for a service provider and service requester to communicate out of band (that is, outside of the normal Web service invocation message exchange) to exchange security credentials. For example, a requester may need to obtain a provider's public key for encryption before sending the message. The WS-Trust specification defines the protocol for assessing and brokering a trusted relationship such as this.

WS-Trust establishes a protocol for:

- Issuing, renewing, and validating security tokens.
- Assessing or brokering trust relationships.

WS-Trust defines the process of exchanging and brokering security tokens so that the requester can obtain the necessary security information to construct trusted messages.

As illustrated in Figure 8-4, WS-Trust defines a SOAP message exchange protocol according to which a service requester and provider can communicate with each other to exchange authentication and authorization credentials.

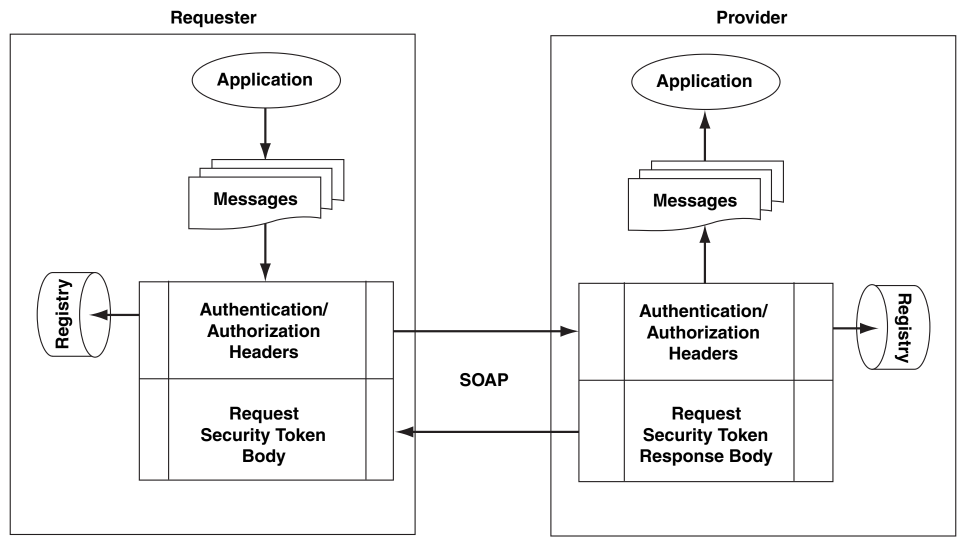
Requester
Provider
Application
Application
Messages
Messages
Registry
Registry
Authentication/ Authorization Headers
Authentication/ Authorization Headers
SOAP
Request Security Token Body
Request Security Token Response Body

Figure 8-4 WS-Trust defines SOAP messages to issue, renew, and validate security tokens.

WS-Trust defines a namespace used to identify messages used to carry out the trust protocol:

```
xmlns:wst="http://schemas.xmlsoap.org/ws/2004/04/trust"
```

The security model defined in WS-Trust is based on a process in which a Web service provider can require that an incoming message from a requester establish a set of credentials (user name/password, key, permission, capability, and so on). If a message arrives without the required credentials, the provider rejects the message with an error. The service provider can assert the credentials it requires using WS-SecurityPolicy. WS-Trust defines the message exchange pattern and format of the credentials information that may need to be exchanged between provider and requester in order to fulfill the policy assertions.

Security tokens are requested using the `<RequestSecurityToken>` message defined in the WS-Trust specification.

Security tokens are returned using the `<RequestSecurityTokenResponse>` message defined in the WS-Trust specification.

WS-SecureConversation

WS-SecureConversation defines a shared security context across multiple message exchanges. It defines a new security context token for the `<wsse:Security>` header block, and it defines a binding for WS-Trust. Instead of having to include the same security credentials in each SOAP message, a provider and requester can use WS-SecureConversation to agree on sharing a common security context. For example:

```
<SecurityContextToken wsu:Id="...">
<wsc:Identifier>...</wsc:Identifier>
</SecurityContextToken>
```

The namespace is as follows:

```
xmlns:wsc="http://schemas.xmlsoap.org/ws/2004/04/security/sc/sct"
```

The context has a unique identifier and can be used to temporarily or persistently store any combination of authentication, authorization, or encryption information that two or more Web services need to share.

As illustrated in Figure 8-5, WS-SecureConversation defines a SOAP message protocol for propagating and establishing common copies of security context at the requester and provider nodes so that they do not have to (for example) exchange authentication information on each Web service invocation request.

The shaded areas indicate shared security context used across multiple message exchanges.

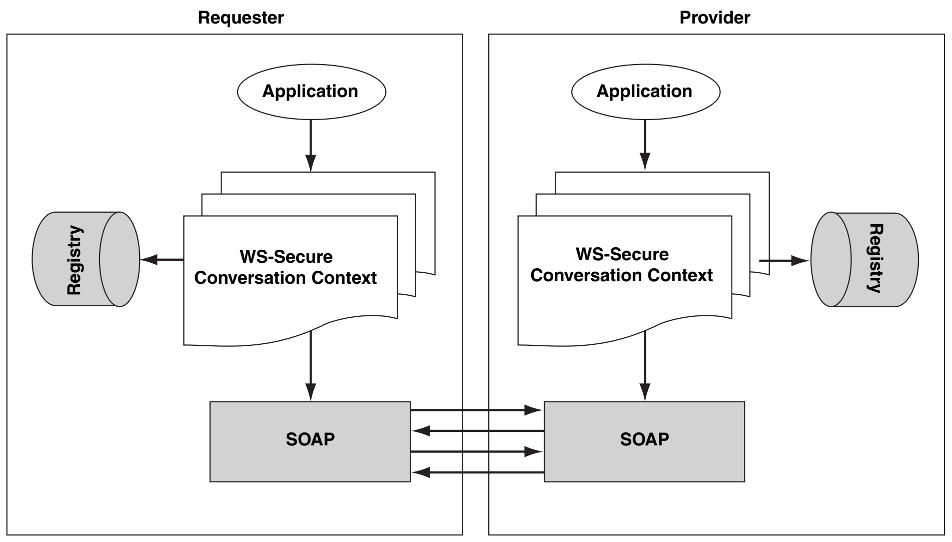

Figure 8-5 Secure conversation establishes shared security context.

WS-Federation

WS-Federation defines how to establish trust relationships across security domains. WS-Trust assumes a single security domain within which the service requester authenticates with the service provider's authentication service. WS-Federation defines a binding of WS-Trust that allows a service provider to accept authentication credentials that come from a different security domain. When an identity is authenticated and access is controlled within a given domain such as an enterprise, department, or execution environment, it may also be necessary to handle these problems for multiple domains because Web services can provide interoperability solutions that cross multiple domains.

The WS-Federation specification defines a message exchange protocol that service requesters and providers can use to establish federation of security domains across multiple trust boundaries. The WS-Federation specification builds on WS-Security, defining a profile of WS-Trust for obtaining and exchanging identity information, a specialized security token service called the Identity Provider (IP), a new policy assertion language syntax called `RelatedService`, and protocols for interacting with attribute and pseudonym services. An attribute service provides the means to obtain information about a principal (that is, an authenticated subject).

A pseudonym service is a specialized attribute service that maintains alternate identity information about a principal. WS-Security, WS-PolicyAssertions, and WS-Trust can be used in combination to accomplish the federation of security trust domains. In other words, WS-Federation takes WS-Trust a step further and establishes a mechanism for exchanging credentials across trust boundaries, not just within a trust boundary.

Accessing services provided on multiple machines or executable software domains may require additional authentication, unless the authentication operations are federated, as they can be for example in a Windows domain via the Windows Active Directory. Like Active Directory for the Windows environment, WS-Federation can be used to support single sign-on protocols for extended security domains. For example, an authentication check can be forwarded to a third party for validation.

The WS-Federation specification defines an integrated model for federating identity, authentication, and authorization across different trust realms. This specification defines how the federation model is applied to active requestors such as SOAP applications. It also defines how pseudonyms can be used to help preserve secrecy when identities need to be protected across domains.

Security tokens are requested using the `<RequestSecurityToken>` message defined in the WS-Trust specification. Security tokens are returned using the `<RequestSecurityTokenResponse>` message defined in the WS-Trust specification. If the requester doesn't already have the WS-Policy for the exchange, it can request it using WS-MetadataExchange.

Security Assertion Markup Language (SAML)

The Security Assertion Markup Language (SAML) from OASIS is an XML application designed to support single sign-on and propagate authorization information. For example, SAML allows a user to log on once to a Web site and then access affiliated Web sites without having to log on again. The affiliated Web sites need to be able to recognize the original user name (or identity). The same mechanism can be used to provide single sign-on for Web services that access different services. The WS-Security SAML profile defines how to use SAML with SOAP, but SAML can be used independently of SOAP and independently of WS-Security, if necessary.

To accomplish single sign-on, SAML defines three basic components: assertions, protocol, and binding. SAML also defines profiles (Browser/Artifact Profile and Browser/POST Profile), which specify how to convey SAML tokens with application requests. Assertions can be one of three types:

- **Authentication**—Validates that the specified subject was authenticated by a particular means at a particular time.
- **Attribute**—A statement by a security authority that supplies qualifying information about the subject.
- **Authorization**—A statement by an authorization authority that grants permission to a subject to perform a specified action on a specified resource.

The protocol defines how applications communicate with a SAML authority to request authentication and authorization decisions. SAML bindings are defined for SOAP and HTTP.

SAML assertions provide security information about subjects, where a subject is an entity (either human or computer) that has an identity in some security domain. A typical subject is a person, identified by user name. A typical assertion conveys information about the authentication of a subject, including any attributes associated with the subjects. The assertion also provides information about authorization decisions that determine whether or not subjects are allowed to access resources.

A SAML assertion is represented using XML and supports nesting so that a single assertion might contain several different internal statements about authentication, authorization, and attributes. (Note that assertions containing authentication statements can carry the results of an authentication that happened previously.) An assertion is issued by a SAML authority, including authentication authorities, attribute authorities, and authorization authorities.

SAML defines a protocol by which requesters can obtain an assertion from a SAML authority, which might be implemented using a security server product such as Netegrity Siteminder of Tivoli Access Manager. A SAML authority can use various sources of information in creating their responses, such as external policy stores and assertions that were received as input in requests. Thus, requesters use the protocol to interact with SAML authorities to obtain assertions, providers use the protocol to interact with SAML authorities to validate assertions, and SAML authorities can be both producers and consumers of assertions.

SAML is different from other security mechanisms because of how it uses assertions about subjects. Other mechanisms rely on a central certificate authority, which naturally raises the issue of trust for the certificate provider. With SAML, any point in the network can assert that it knows the identity of a user or piece of data. It is up to the receiving application to accept whether or not it trusts the assertion, which sometimes will mean that additional authentication information is needed.

When SAML assertions are used with WS-Security, they can be referenced using the `<wsse:SecurityTokenReference>` element. SAML assertions can also be placed directly inside the `<wsse:Security>` header block. When using the token reference, the `<saml:Assertion>` element is not embedded in the `<wsse:Security>` header. SAML assertions take the format of `<saml:Assertion>` and typically start with a UUID. The remainder of the information is typical SAML information, including information about the SAML issuer. The Web service receiving the SAML assertion can find the assertion issuer and check the assertion.

Use SAML on Its Own or with WS-Security?

This question naturally arises when a technology is available both as a standalone, independent technology and as an integral part of another technology. As a general rule of thumb, the answer is typically determined by the amount of coding necessary to accomplish the task and by the degree to which interoperability can be assured. Whenever a technology such as SAML is profiled inside another technology such as WS-Security, conforming implementations of WS-Security are required to support SAML (assuming the conformance includes the SAML profile, of course). If you are using a Web services platform or a set of Web services products that support WS-Security, the simplest and most interoperable choice is to use SAML within WS-Security. If all you require is SAML, on the other hand, it may make more sense to simply use SAML directly and require services in your platform (and your trading partners' platforms, if any) to also support SAML.

An authentication assertion identifies the subject (using a `NameIdentifier` and/or a `SubjectConfirmation`), and it contains an authentication statement that specifies when and how the subject was authenticated. Role information may be associated with the subject using an attribute statement. Authorization information may be associated with the subject using an authorization statement. All three types of assertion statements may be included in a single `<saml:Assertion>` element, but they are still three different types of statements.

SAML assertions can also have version numbers and signatures. SAML assertions can also specify condition elements for credential expiration dates. SAML defines a protocol and behavior of the assertion providers. SAML requires SSL certificates to provide digital signing and encryption of SAML assertions.

SAML can provide protection from replay attacks by requiring the use of SSL encryption when transmitting assertions and messages. Additionally, SAML provides a digital signature mechanism that enables the assertion to have a validity time range to prevent assertions from being replayed later.

XACML: Communicating Policy Information

The Extensible Access Control Markup Language (XACML) is an XML application for writing access control polices.

Access control security mechanisms have two sides: the side that performs the check to see whether a user is authorized to access the Web service, and the side that defines and manages the information that the access control mechanism checks. In other words, the access control information needs to be defined in order for it to be checked.

The XACML specification provides an access control language to define access policies and a request/response protocol to request authorization decisions. XACML can also be used to connect disparate access control policy engines.

XACML defines a set of rules for the XML encoding of what data a person is allowed to read. For example, it could define which HR records you can access from the HR Web site based on whether you are the employee, the authorized parent or guardian of the person in the HR records, or the physician or other authorized HR agent who can update the records.

XML Key Management Specification (XKMS)

XKMS is an XML-based mechanism for managing the Public Key Infrastructure (PKI).

PKI uses public-key cryptography for encrypting, signing, authorizing, and verifying the authenticity of information, including Web services messages. Public and private keys can be used in XML Encryption and XML Signature, for example, and to provide additional levels of authentication for an HTTP connection.

The XKMS specification defines a set of Web services to manage the task of key registration and validation, based on the use of a third-party “trust” utility that manages public and private key pairs and other PKI details. In other words, XKMS defines Web service interfaces to key management systems so that Web service applications can access and use their facilities. Otherwise, key management requires a manual process of generating keys, placing them in their proper directories, and publishing their location.

XKMS works with any PKI system, such as those provided by Verisign and Entrust, passing the information back and forth between it and the Web service. XKMS is a W3C specification.

Essential to the public/private key mechanism is the ability to manage and distribute key pairs. If a third party generates the associated key pairs, a management facility such as XKMS is necessary to ensure that the right keys end up in the right place.

Data-Level Security

XML Signature and XML Encryption are fundamental security specifications for protecting Web services data. Because Web services specifications are all applications of XML, the specifications themselves can be protected using these core XML technologies. For example, if you want to protect your WSDL files against unauthorized access, you can encrypt them. If you want to protect your WSDL files against tampering, you can sign them.

These specifications, along with SAML, XACML, and XKMS, are not specific to Web services because they are general to XML and are not specifically adapted to SOAP and WSDL the way the other specifications in this chapter are.

XML Signature defines how to verify that the contents of an XML document have not been tampered with and arrived unchanged from the way they were sent. XML Encryption describes how to encrypt all or part of any XML document so that only the intended recipient can understand it.

It's especially important to consider using these XML security technologies when the XML data needs to be protected outside the context of a SOAP message and when the Web services metadata needs to be protected from unauthorized access. WS-Security uses XML Signature and XML Encryption to help ensure confidentiality and integrity of SOAP messages, but it does not describe how to use these XML technologies outside the context of SOAP and WSDL, which may be important for some applications, especially those storing XML in a kind of intermediate format between transmissions. If a purchase order (PO) has to be stored in the middle of a business process, for example, XML

Encryption can be used to guard against unauthorized access to its contents, and XML Signature can be used by the next step in the business process to ensure that the PO has not been tampered with.

XML Encryption

Encryption of the XML payload when carried over HTTP can be accomplished using SSL, but sometimes that's not enough. When carrying XML over other transports, potentially over multiple transports, or when storing XML documents in a file or in a database, it is helpful or even necessary to have a specific mechanism for encrypting the XML documents.

When encrypting an XML element or element content, the `EncryptedData` element defined in the XML Encryption specification replaces the element or content (respectively) in the encrypted version of the XML document. As with many things related to XML, encryption works at any level of nesting. Either the entire document (except the encryption headers) or any element within it can be encrypted.

Selective encryption is useful when only part of a document needs to be kept private. It's possible to encrypt the tags as well as the data so that no one can see what the data is supposed to contain, such as hiding a `<creditcard>` tag within a `<CipherData>` tag.

For example:

```
<?xml version='1.0'?>
  <PaymentInfo xmlns='http://www.iona.com/artix/paymentService'>
    <Name>Eric Newcomer</Name>
    <CreditCard Limit='50,000' Currency='USD'>
      <Number>5555 5555 5555 5555</Number>
      <Issuer>Example Bank</Issuer>
      <Expiration>04/02</Expiration>
    </CreditCard>
  </PaymentInfo>

<?xml version='1.0'?>
  <PaymentInfo xmlns='http://www.iona.com/artix//paymentService'>
    <Name>Eric Newcomer</Name>
    <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <CipherData>
      <CipherValue>A23B45C56...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

This example illustrates both plain and encrypted versions of the same data. Encrypted data is contained with the `CipherData` element. If an application requires all information to be encrypted, the whole document can be encrypted as an octet sequence. This applies to arbitrary data including XML documents. For example:

```
<?xml version='1.0'?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
 MimeType='text/xml'>
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
  </CipherData>
</EncryptedData>
```

The `<EncryptedData>` element can't be nested, but an `<EncryptedData>` tag can be used at the same level as another `<EncryptedData>` tag, causing already encrypted data to be encrypted again. This is convenient for developers who don't want to worry about the presence of another `<EncryptedData>` tag in the documents they're encrypting.

The `EncryptionMethod` is typically a secret key mechanism such as triple DES or RC4, or sometimes an RSA public key or similar algorithm, depending on the level of protection required.

A reference list contains all encrypted items within the document. A URI can be used to point to the encrypted data.

XML Signature

XML Signature[5] ensures that the provider knows that the part(s) of the document that have been signed haven't been changed between the time it was sent and received. The receiving application (such as a Web service provider) has no obligation to understand what's been signed, but if it can understand the signed

[5] XML Signature was developed jointly by the W3C and the IETF (RFC 2807, RFC 3275).

part of the document, it can use the signature to determine whether that part's contents are unaltered and to authenticate the document's author. Applications can sign multiple data objects, some of which may not be XML.

An XML Signature may be applied to the content of one or more parts within an XML document. Because XML documents can contain or reference binary objects and multimedia types, XML Signature has been designed to support those types of objects in addition to XML elements and attribtues.

A signed object is guaranteed by the presence of the signature either to be unaltered or to provide a mechanism for the receiver to determine whether or not the signed object has been tampered with.

The XML Signature associates a key with referenced data objects but does not specify how keys are associated with persons or institutions,[6] nor the meaning of the data being referenced and signed. Key management for XML Signature, as for other aspects of key-based security, is assumed to be handled by another technology, such as a key registry, XKMS application, or other directory service.

The data objects are canonicalized and digested before being sent. Digesting runs a hash algorithm over the data object, and canonicalization removes all white space and formats the document according to the canonicalization algorithm. You must canonicalize the data before signing it to ensure that you get the same results each time. Then you digest it and sign the digest:

```
<Signature ID?>
     <SignedInfo>
       <CanonicalizationMethod/>
       <SignatureMethod/>
       (<Reference URI? >
         (<Transforms>)?
         <DigestMethod>
         <DigestValue>
       </Reference>)+
     </SignedInfo>
     <SignatureValue>
    (<KeyInfo>)?
    (<Object ID?>)*
   </Signature>
```

[6] Which is why WS-Security may be needed.

This example from the XML Signature specification illustrates the XML Signature syntax structure. The `<CanonicalizationMethod>` tag identifies the mechanism used for distilling the information.

Signatures are associated with data objects using URIs. Within an XML document, signatures are related to local data objects via fragment identifiers. Such local data can be included within a signature or can enclose a signature.

The specification defines a schema for capturing the result of a digital signature operation applied to arbitrary data. XML signatures add authentication, data integrity, and support for non-repudiation to the signed data.

XML Signature can be used to sign only specific portions of the XML tree rather than the complete document. This is important when a single XML document needs to be signed multiple times by a single or multiple parties. This flexibility can ensure the integrity of certain portions of an XML document, while leaving open the possibility for other portions of the document to change. Signature validation mandates that the data object that was signed be accessible to the party interested in the transaction. The XML signature must indicate the location of the original signed object.

Summary

Security is a complex field awash in technologies and protocols to meet an ever-growing series of threats to data and programs. Protecting data against unwanted access typically involves encrypting Web services messages, and a variety of options exist for doing so. It's important when selecting an option to determine compatibility with the services you're interacting with, and to ensure that the overall SOA supports a consistent technology, or set of technologies. Often, more than one encryption technology is needed to handle the variety of services arriving from a variety of sources in an SOA, and mechanisms are available for this purpose.

Protecting against unwanted access to programs and IT resources involves using potentially strong authentication techniques combined with authorization checks to restrict access to only those who need it. Again, a variety of technologies exist for authentication, and picking the right one or set is important for the smooth and efficient functioning of an SOA. When exposing services externally, it may be necessary to support a choice of authentication mechanisms for different consumers.

Whenever decisions are made concerning the selection of the most appropriate security technology, it's important to codify and formalize them in policies. A good security solution starts with a well-reasoned and thoroughly researched statement of policy. Web services provide mechanisms for expressing those policies in machine-readable form, but it's important to thoroughly document the overall security policy and the threats it's designed to guard against.

With security, it's easy to think that you never have enough, but striking the right balance is important because each security technology comes with a built-in performance penalty. The stronger the encryption, the more processing power it takes to encrypt and decrypt, for example. Use only as much security as you really need.