

# **Web Services Security**

Today's application environment is a complex web of networks, servers, routing, and processing intermediaries. More than ever, concern must be taken for the protection of critical data. An organization might be legally bound to protect the confidentiality of its customers' private information, and might need to ensure the integrity and origin of data that it receives. This chapter will introduce the core of Data Power's cryptographic capabilities. We will describe the vulnerabilities of distributed applications and offer solutions to protect valuable assets both within the application and the client environments.

## **Web Services Security**

We can assume that because you are a reader of this book, you have more than a passing interest in the world of Web services and security. You might have heard of the WS-Security specification and may, in fact, assume that WS-Security is synonymous with Web services security. This is not the case! The remainder of this chapter makes this clear.

Security of Web services may be thought of as existing on two primary axes; first there is the transport level where point-to-point security is offered. Secondly there is the end-to-end security offered by cryptographic techniques such as digital signatures and encryption. There will be situations where either or both of these techniques will provide the security that is required for your application and other situations where you will be required to use a specific technique either due to policy or practical issues of data protection. The technique you use will affect not only the integrity and confidentiality of your data, but could also have performance implications. We will see how DataPower addresses these concerns.

The following sections present some fundamental information regarding message vulnerabilities, security goals, terms and basic definitions, and the relative strengths and weaknesses of various Web services security technologies and WS-Security in detail. Examples will be presented to demonstrate these principles on the DataPower device.

## Message Exchange and Vulnerabilities

Common application message exchange (particularly within distributed systems) involves the transmission of data through many intermediaries. Often these intermediaries are unknown or are out of the control of the sender or recipient. A client might enter data within a web browser for consumption by an application server, or a business might transmit a well-formed XML document to a subsidiary outside of the control of its internal network. Often complex network infrastructures require the routing of application traffic via dynamic paths, further complicating the determination of where message data will travel. In either case, this transmission could potentially expose the data to inspection and potential manipulation by unscrupulous parties.

As the saying goes, “The best-laid plans of mice and men often go awry” and IT technologies are not going to do a bit of good if they are not backed up with qualified processes. So, before we jump into the world of security, let us remind ourselves that no implementation of DataPower security functionality is going to solve your problems if it is not backed up by a secured IT infrastructure. All the cryptography in the world will do you no good if you leave the administrative password unchanged. You have no chance of protecting yourselves from exploitation if you do not perform reviews and audits of configuration changes and do not administer the audit log that the device produces. There’s information on these topics in other chapters, such as Chapter 12, “Device Administration,” Chapter 14, “Logging and Monitoring,” and Chapter 15, “Build and Deploy Techniques.” However, you must understand that security is not a topic that exists in a vacuum; it is affected by outside influences that you must prepare for and protect against holistically.

## Integrity

When a message is sent using the postal system, the contents are typically enclosed within a sealed envelope. The recipient of that message makes the assumption that the message was not exposed or altered during transit. This might, of course, be a misguided assumption because there is no guarantee that the letter was not intercepted and its contents altered prior to its delivery. Message integrity has the goal of providing that guarantee. Integrity’s goal in the world of digital data transmission is to provide a mechanism and process such that the recipient might validate that the message has not been changed since it was sent by its creator. This goal does not guarantee that the message was kept private; simply that it was not altered.

## Confidentiality

Referring again to our postal system example, the recipient has no guarantee that the message contained within the envelope delivered to her has not been altered. Furthermore, it is equally obvious that she has no guarantee that the message has not been viewed. Perhaps an unscrupulous intermediary has gone to her mailbox, opened the envelope, read the enclosed message, and resealed the

envelope or replaced it with another envelope entirely. Who's to know? The most typical and oldest methods of ensuring that the enclosed message remains private is to either keep it out of the reach of all but intended parties (certainly unlikely in our mail or network examples) or to transform the message into an encrypted format that is unreadable by all but the intended party.

With modern multiply-connected systems, you might want to pass information through one party to another, and keep some of the content confidential from the first party. Consider a situation where a customer is ordering something from an online store and needs to pass his credit card information. The online store does not need to know any of the credit card data; the store just needs to pass it to a bank to verify that the charge is authorized. Passing information such as this through one party to be read only by another party is part of the confidentiality capability of Web services security.

## **Nonrepudiation**

We are all familiar with the significance and intent of a handwritten signature. It implies that the signatory has placed his oath of compliance on the document to which it is bound. One might, in fact, be held to account for this signature in a court of law and be compelled to abide to its intended obligations unless the signature is proven to be illegitimate. For example, it may be illegitimate if it was forged or forced. The goal of nonrepudiation in the digital world is the same; the recipient of a message might need to ascertain with a high degree of reliability that the message came from the indicated originator and have a mechanism to assert that origin.

## **Authentication, Authorization, and Auditing**

When someone asks you for something you possess, you might take quite different actions based on who is making the request, and what she asks for. If while standing on a subway platform, a stranger asks you for the time, you would most likely provide her with a response if you had the capacity, such as you have a watch and you speak the same language. However, if the same individual asked for your Social Security number, you would hopefully respond quite differently.

The goal of Authentication, Authorization, and Auditing (AAA) is, of course, to identify requestors, the resources they desire, deciding whether they are entitled to it, and recording the exchange. You can read all about this subject in Chapter 16, "AAA," and Chapter 17, "Advanced AAA," but for the purposes of our Web services security discussion, we will be interested in how this identity and other meta-data is transmitted from requestor to provider.

## **Cryptographic Terminology, Basic Definitions**

The encryption of messages has been around for a long time. It is said that in the fifth century BCE, secured messages were tattooed onto the scalp of the messenger and only sent when the hair had regrown. Certainly an inadequate timeframe for today's millisecond commerce! The discussion to follow references key elements of cryptography. There is a close relationship between this discussion and the discussion presented in Chapter 18, "DataPower and SSL," as the technologies share many key components. The terminology introductions here will be brief and you might want to revisit the Secured Sockets Layer (SSL) discussion for a more in-depth analysis.

## Public Key Infrastructure

In the digital world, public and private keys are often employed to perform cryptographic operations, such as encryption of message data. The use of key pairs (public/private) is known as asymmetric encryption. It is vital that the private key is protected, while its public counterpart, the public key (often carried in a certificate), can be freely distributed. Certificates are typically validated by a Certificate Authority, and that authority might revoke a previously distributed certificate.

Symmetric encryption requires that a shared secret key is possessed by both parties. Asymmetric encryption utilizes public and private key pairs that do not require predistribution. Although symmetric encryption requires the distribution of the shared key prior to message encryption or decryption, it is far more efficient than asymmetric encryption.

Chapters 12 and 18 discuss some of the aspects of key distribution within the DataPower environment. Revocation lists may be maintained so that when certificates are revoked, you are aware of it, and you no longer accept them. Additionally, when you produce or request certificates, you need to ensure that the validity period is not too excessive and that the cipher strengths meet your security policies.

## Canonicalization

XML is flexible in the authoring of documents; it is possible for a given XML document to be represented in multiple physical structures. As seen in Listing 19-1, the two documents while written in differing formats (attributes ordered differently, whitespaces differ) are equivalent in their XML representation. That is to say that when consumed by the XML Path Language (XPath) or XML parsers, they are processed similarly, and client code would be unable to distinguish between them.

### Listing 19-1 Structurally Unique yet Logically Equivalent XML Documents

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <fee name="fee" type="element">fee</fee>
</doc>
-----
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <fee type="element" name="fee" >fee</fee>
</ doc>
```

If these two documents were processed in the textual representation presented in Listing 19-1 by cryptographic algorithms, the results would be quite different. To get around this potential problem, a transformation of the original document, known as canonicalization, is performed. Canonicalization (a subject in its own right) basically describes the transformations necessary to normalize documents into a standardized format. With XML documents, this process describes the ordering of attributes, the handling of whitespace, the representation of empty elements, and other characteristics of XML documents that might account for differing physical representation of logically equivalent documents. Prior to performing the hash calculation, an XML document is typically formatted by the canonicalization method to avoid structural issues that would cause document integrity checking to fail.

## Digital Signatures

Digital Signatures can ensure the integrity of messages by attaching a cryptographically generated hash of the original message (or parts thereof) to the document. Public and private keys are often used for this process. XML Digital Signatures (XMLDSIG), a joint effort of the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF), provides the specification for the implementation of digital signatures and the structure of the resultant XML document format. OASIS (Organization for the Advancement of Structured Information Standards) provides similar specifications for WS-Security in conjunction with SOAP documents. XMLDSIG can be used to sign entire XML documents, selected XML elements, as well as other types such as binary images or Multipurpose Internet Mail Extensions (MIME) attachments. All document types result in similar encapsulations. XMLDSIG also plays an important role in many other security techniques and is vital in the integrity processing of these protocols.

By associating a digital signature with a document, the intent is that the holder of the private key used to generate the signature has placed its mark on the origin of the document. The verification of this signature proves that the signature was generated by the holder of the private key and (assuming it has not been stolen or otherwise compromised), its origin may not be repudiated. Similar to the handwritten signature, a digital signature might also be enforced under certain conditions in a court of law and might be held to similar legal standards such as whether the signer intended to sign the document and had the right to sign it.

The reader may want to consult the XMLDSIG protocol for definitive implementation information, but the following discussion describes the basis for XMLDSIG generation and validation.

Typically a hash (digest of canonicalization result) is calculated representing the signature target (be it an XML Document, XML Node, attachment, image, binary data, or combination thereof). This creates a value that corresponds to the original content and is unique to that content. A hash mechanism such as the Secured Hash Algorithm (SHA1) is processed against the byte stream and produces a fixed length string (20 bytes in the case of SHA1.) This provides a more condensed basis for further cryptographic processing. It is computationally infeasible for the message to be reproduced from the resultant string, and no two messages *should* produce the same digest value. There exist arguments exposing potential rebuttals to these assumptions for certain hashing algorithms; however, given constraints such as validity periods (limiting the lifetime of the signature) the computationally infeasible argument has significant merit.

Having produced the digest value, it is encrypted using one of the public/private key pairs. In the case of XMLDSIG, the private key of the sender is used, allowing verification of the signature via the freely available public key.

The validation process involves the recipient producing a digest value of its own using the same hash algorithm as the sender. If it is equivalent to the sender's version, the message has not been modified in transit. Decrypting the encrypted version of the digest using the sender's public key, and validating that key using the Public key infrastructure (PKI) mechanisms ensures that the identity of the sender is validated.

## Encryption

Encryption ensures the confidentiality of messages (plaintext) through the implementation of a reversible transformation algorithm to produce encrypted ciphertext. If the message is intercepted prior to its ultimate destination, the contents of the message will not be exposed. However, this methodology does not preclude a man in the middle from changing the contents of the message, encrypted or not. For this reason, it is advisable to combine signatures with encrypted messages to ensure that even though the original contents are not exposed, the message was not altered in transit.

### **TIP—FOR ADDED SECURITY, FIRST SIGN THE MESSAGE, AND THEN ENCRYPT**

Signing the message ensures the integrity of the original message content. Encryption ensures confidentiality and the integrity of the signature metadata.

As has been discussed, two primary types of encryption algorithms exist: symmetric, where a single, shared secret key is used for both the encryption and decryption operations, and asymmetric, in which a public/private key pair is utilized. Symmetric encryption is orders of magnitude faster than asymmetric; however, the difficulty lies in the management of the keys. Symmetric keys must be shared out of band prior to the decryption process. Asymmetric cryptography resolves key management by the free distribution of public keys.

Encryption is performed using the public key of the intended recipient. This ensures that only the holder of the private key can decrypt it. DataPower (as will be examined later in this chapter) enforces the proper use of keys in all encryption and decryption operations, and digital signatures for that matter. In an effort to ameliorate the asymmetric performance problems, encryption systems typically employ the public/private key pair for asymmetric encryption of a dynamically generated shared secret symmetric key (session key) that is used in the actual message encryption process. This is quite similar to the ephemeral key generation processed used in SSL, you might want to again refer to Chapter 18 for a description of this process.

The XML Encryption (XMLENC) protocol specification is defined by the World Wide Web Consortium (W3C). It defines the XML structure that results from the encryption process and describes the algorithms and keys that are used. XMLENC can be used to encrypt any type of document such as XML, binary images or attachments that all result in similar encapsulations. Individual fields can be encrypted or the entire document can be encrypted, and multipass cryptography (super encryption) may be employed.

You might want to consult the XMLENC protocol for definitive implementation information, but the following describes the basis for XMLENC generation and validation.

As was the case with Digital Signature generation, the first step in encryption is the canonicalization of the target data. A canonicalization transformation converts the plaintext into a consistent physical structure and normalizes elements (in the case of XML), whitespace, and other

structurally variable document components. The resulting byte stream is then encrypted with a symmetric (session) key. The symmetric key used is then encrypted with the public key from the intended recipient. Decryption is simply the reverse; the session key is decrypted using the recipient's private key and then used to decrypt the ciphertext into plaintext.

## **SSL/TLS (HTTPS) Use and Vulnerabilities**

SSL and associated protocols, such as Transaction Layer Security (TLS), provide transport layer security and are often employed to facilitate the goals of integrity and confidentiality. As is discussed in Chapter 18, HTTPS (SSL over HTTP) is employed for these purposes. SSL employs a combination of asymmetric and symmetric cryptography. Public/private key pairs are used to asymmetrically encrypt a dynamically generated session (ephemeral) key that is used for the communication's data encryption. By validating the public keys of the SSL communication peer (the client or server), the identity of the peer can be determined, and the message can be encrypted and protected from third-party exposure.

However SSL/TLS is not always sufficient for complete data protection because it is only effective point-to-point (between communication endpoints.) If, for example, an SSL terminator is used in a network prior to the delivery of the message to its intended recipient, the message is decrypted into plaintext. This might expose vulnerabilities in several ways; log messages might print the data to external devices (with private information), and importantly, the data is transmitted between these endpoints (and potentially others) in its original form and exposed to network sniffers and other unscrupulous methods of inspection. Similar vulnerabilities exist when messages are transmitted across trust zones where they could be intercepted by unintended intermediaries.

Partial message encryption is also unavailable with SSL because it encrypts the entire message and cannot encrypt only certain parts. There might be occasions when the routing of a message is dependent on an element within the message body. Exposing the routing information necessitates decrypting and exposing the entire message body. Or for example, a credit card number might be part of a request sent over the Internet, and might only be intended to be decrypted by a credit card processing service, while address information may be viewable by others. It would be beneficial in this instance to be able to decrypt only those message components that a particular intermediary requires and leave the remainder of the message protected.

There are many instances when SSL will meet the needs of a particular application, and in those cases it is all that is required. SSL is easily established in most environments. Applications hosted on WebSphere Application Server for example may be configured to utilize SSL with ease. The DataPower devices are also, of course, well suited to receive SSL and to validate either SSL clients or servers.

## **Web Services Security**

Web Services security might be thought of as transport layer security plus additional techniques and methodologies. It's not a standard, but more of a concept. The goals of Web Services security are the core group of features we've discussed; confidentiality, integrity, nonrepudiation and

authentication and authorization for resource access. Protocol headers (such as those used for basic authentication) are frequently used to convey identity information. However the lack of confidentiality with these techniques requires the concurrent use of transport layer security. This problem has been addressed with proprietary solutions such as the encrypted Lightweight Third-Party Authentication (LTPA) token utilized by applications within the WebSphere environment; however, this does restrict their usage to supporting environments.

Web Services security is lighter weight than WS-Security with its greater emphasis on extensibility protocols and heavy use of XML for policy definition. And many times the more traditional Web Services security might be all that is required.

Transport level security provides not only point-to-point security and the ability to encrypt traffic but can also authenticate (though not authorize to the resource level) the endpoints, or peer via the inspection of the clients' certificate. XMLDSIG and XMLENC can be used to provide integrity checking and confidentiality.

Additional techniques such as Access Control Lists (ACLs) can also be used to assist in secured client access. However, with the use of Dynamic Host Configuration Protocol (DHCP) this technique might be limited to consistently addressed endpoints such as in-house resources.

### **DataPower Web Service Security Features**

DataPower supports a wide range of traditional security features. SSL/TLS is fully supported. Encryption and signature support is available for raw XML and SOAP document as are ACLs. The AAA functionality described in Chapter 16 and Chapter 17 describes methods to extract SSL credentials from SSL certificates and provides extensive token translation support. Of particular interest to users of WebSphere is the ability to create LTPA tokens for SSO across an application cell. This is presented in Chapter 21, "Security Integration with WebSphere Application Server." Support for the Security Assertion Markup Language (SAML) is also provided (as discussed in Chapter 16) extending the trust domain across organization boundaries and providing the implementation of Single Sign On (SSO) solutions.

### **WS-Security**

WS-Security is not a protocol or specification in and of itself. Rather it was initiated to solve more complex problems than that offered by the point-to-point solutions of Web services and transport layer security, by leveraging existing standards and protocols. Messages may be sent over unanticipated protocols and across trust domains with requirements such as preserving identity information. Multiple encryption and decryption destinations might be required to individually expose message data to only the intended recipient.

WS-Security facilitates these goals by utilizing a SOAP Header to carry security-related information. For example, if XMLDSIG is used to generate a signature, or XMLENC to encrypt an element, the SOAP Header is used to carry the key and algorithm information.

WS-Security also identifies a mechanism for communicating identity information. The Username Token profile stores this data within the SOAP Header, and the Binary Security Token profile defines a mechanism for carrying crypto tokens such as Kerberos tickets or X.509 certificates. You



can read more about these specifications on the Organization for the Advancement of Structured Information Standards (OASIS) Web site, and in Chapter 16 and Chapter 17.

WS-Security is not an island, and this framework is constantly being enhanced with additional supporting specifications. We'll talk about WS-Policy (which is used to define the requirements of a Web service) later in this chapter, but there are many others such as WS-Trust for the definition of trust relationships across domains, WS-SecureConversation for dynamic security decisions, WS-ReliableMessaging for the assured delivery of message data, WS-Addressing for dissemination of routing information, WS-Interoperability and WS-Basic Profile for the definition of standards for the sharing of information describing Web Services and how they should interoperate, and WS-Federation for the federation of identity information between parties involved in multisite transactions. Again, you can read about the latest WS-Security specifications (often referred to as WS-\*) on the OASIS Web site.

## DataPower WS-Security Features

One of the criticisms of WS-Security is the performance hit and complication involved in its implementation. That is where DataPower really shines. Not only does it provide support for an ever expanding list of specifications, but through the use of its purpose built firmware and cryptographic hardware accelerators, it does so at a speed and efficiency of configuration that makes their implementation viable and effective. We'll see an example of WS-Policy implementation on DataPower later in this chapter; you'll see how specifications such as WS-SecureConversation are implemented along with more typical security patterns. DataPower also provides support for WS-Addressing and WS-ReliableMessaging through the Multi-Protocol Gateway and Web Service Proxy Services, and WS-Trust through AAA policies.

DataPower also provides built-in Replay filters that may be added to the multistep policy via the Filter action to check for the repeated submission of like messages (Replay Attacks). This comparison can be performed using the WS-Security Password Digest Nonce or the WS-Addressing Message ID. These selections are made from the advanced screen. The filter will reject subsequent messages with similar characteristics within the time frame specified.

## WS-Policy and Policy Governance

If you have read Chapter 10, "Web Service Proxy," or have worked with Web services and WSDL definitions, you understand the power of the WSDL to communicate the structural and networking requirements of a Web service. By consuming the WSDL, the client can develop messages (either automatically or via visual inspection) and understand where and how to send them to invoke the service.

WS-Policy takes the same approach in the specification of Web service interoperability, defining how Web services must be secured, whether requests must be submitted over particular protocols, or whether the request must be part of a transaction. The WS-Policy Framework and WS-Policy Attachment specifications define the metadata describing these requirements, which (in a similar fashion to the WSDL) might be automatically consumed or visually inspected by clients for the proper delivery of service requests.

Assertions are used to define domains such as security, messaging, transaction control, and reliability. Through the definition of collections of assertions (patterns) within these domains, policies may be defined describing the restrictions required for a particular service. There are several existing policy domains such as Web Service Security Policy, Web Service Reliable Messaging Policy, and Web Service Atomic Transaction. These policies may be authored within the WSDL document itself or published in repositories such as Universal Description Discovery and Integration (UDDI) or WebSphere Service Registry and Repository (WSRR). Listing 19-2 demonstrates a policy expression containing two policy assertions that defines the requirements of the signing and encrypting of the message body.

### Listing 19-2 Sample Policy Expression and Assertions

```
<wsp:Policy wsu:Id="sign-before-encrypt-1">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
      <sp:EncryptedParts>
        <sp:Body/>
      </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

## DataPower WS-Policy Features

DataPower supports the utilization of WS-Policy assertions within the definition of policy configuration. External Policy (not from the WSDL) may be attached at various levels within the WSDL policy definition of the WSP (Web Service Proxy), and policy that has been authored within the WSDL may be attached as well. DataPower has encapsulated many of the aforementioned domain patterns within the /store/policy/templates directory of its file system for ease of implementation.

Policy assertions contain the requirements of a service such as dictating the use of HTTPS in a Transport Binding assertion, but do not define the keys and certificates this protocol would use. DataPower defines Parameter Set for this purpose, and provides screens for their assignment to bindings. We demonstrate the implementation of WS-Policy later in this chapter.

## Digital Signatures on DataPower

Having described the reasons for implementing digital signatures, we turn our attention to their implementation on the DataPower device. DataPower provides a simple mechanism for the signing and verification of message data. The multistep policy editor provides Sign and Verify Action icons, which might be placed onto the policy in any location. Standard use cases are typically handled with little effort other than assignment of key information, whereas more atypical requirements can be managed through fine-grained parameter assignment available via the Advanced tab of the Sign

action. DataPower can be used to sign and verify signatures of various types of data such as XML documents, SOAP documents, as well as attachments such as MIME and Direct Internet Message Encapsulation (DIME). The following examples demonstrate three typical uses: document level signing of a SOAP document, selective field level signing, and the signing of a MIME attachment.

## Document Level Signing

Signing an entire document is easily performed using the Sign action. As can be seen in Figure 19-1, the only configuration detail required is the key and certificate to be used. The signer's private key is used to generate the signature via the encryption of the digest produced by the message target. The public key is specified on this action so that the recipient can verify the signature after verifying the validity of its public key via their PKI infrastructure.

**DATAPOWER X150** Help

**Configure Sign Action**

**Basic** Advanced

---

**Input**

Input: (auto) (auto) ▼

---

**Options**

**Sign**

**Envelope Method**

- ☐ Enveloped Method
- ☐ Enveloping Method
- ☐ SOAPSec Method
- ☒ WSSEC Method
- ☐ Advanced
- \*

**Message Type**

- ☒ SOAP Message
- ☐ SOAP With Attachments
- ☐ Raw XML Document
- ☐ Selected Elements (Field-Level)
- ☐ Advanced
- \*

**Asynchronous** ☐ on ☒ off

**Key** BookPurchaseClient ▼ + ... ☒ Save

**Certificate** BookPurchaseClient ▼ + ... ☒ Save

---

**Output**

Output: signedBookPurchase (auto) ▼

Delete Done Cancel

**Figure 19-1** Sign action with WS-Security (WSSEC) method and SOAP Document type selected.

There are four different envelope methods available on the Sign action that determine how the signature is carried within the resulting message. Two options may be used with raw XML without a SOAP envelope. Enveloped and enveloping describe the placement of the signature and elements used to contain it. The more common use case (especially with Web services) is the use of the WS-Security (WSSec) or the SOAPSec methods. You should refer to the XMLDSIG Core specification for definitive information on signature encoding.

Listing 19-3 shows the sample SOAP document used for the following examples. This is a simple book purchase request, and it contains some information such as credit card data that should be considered sensitive.

### Listing 19-3 Sample SOAP Document

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:exm="http://www.example.org/BookService/">
  <SOAP-ENV:Body>
    <exm:BookPurchase>
      <isbn>0321228006</isbn>
      <price>19.95</price>
      <creditCardInfo>1234567890121234-0808</creditCardInfo>
      <cardHolderName>Ishmael</cardHolderName>
      <address>Sailing Vessel Pequod, Old Harbor, Gloucester MA
01111</address>
    </exm:BookPurchase>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 19-4 shows the results of the signing action. Several of the cryptographic fields such as digest values and X.509 certificates have been truncated for brevity. There are many new fields added that describe the processes required to generate and to verify the signature.

First a SOAP-SEC:Signature header was added to the SOAP:Header. All the details of the procedure are carried within the Signature element. The actual signature is contained within the SignatureValue in Base64-encoded format.

To determine how the signature was created, you need to understand the details of the SignedInfo element. It contains the canonicalization method used to create the byte stream from the original message, and the signature method (for example Kerberos or the RSA algorithm) used to create the signature.

The XMLDSIG specification allows for the signing of multiple fields, and the Reference element is used to describe each occurrence. In our document level signature example there is only one. Each of the potentially multiple entries contains a pointer to the signed content that could not only reference an element within the current document, but a URL reference to any network accessible data. Here we indicate that the body of the SOAP envelope has been signed. The Reference element also contains a digest of the content, how the digest was calculated, and transformations that are required prior to generating the digest, which in our example is XML canonicalization.

Finally, the public certificate of the signer is carried within the `KeyInfo` element; it will be extracted and validated by the recipient and used for signature verification. It is assumed that this certificate will be validated by the recipient to maintain the chain of trust.

#### Listing 19-4 Signed XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:exm="http://www.example.org/BookService/">
  <SOAP:Header xmlns:SOAP-
SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-SEC:Signature SOAP:mustUnderstand="1">
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
          <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <Reference URI="#Body">
            <Transforms>
              <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-
c14n-20010315"/>
            </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>(Truncated)</DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>(Truncated)</SignatureValue>
        <KeyInfo>
          <X509Data>
            <X509Certificate>(Truncated)</X509Certificate>
            <X509IssuerSerial>
              <X509IssuerName>CN=BookPurchaseClient</X509IssuerName>
              <X509SerialNumber>1737560841</X509SerialNumber>
            </X509IssuerSerial>
          </X509Data>
        </KeyInfo>
      </Signature>
    </SOAP-SEC:Signature>
  </SOAP:Header>
  <SOAP-ENV:Body id="Body">
    <exm:BookPurchase>
      <isbn>0321228006</isbn>
      <price>19.95</price>
      <creditCardInfo>1234567890121234-0808</creditCardInfo>
      <cardHolderName>Ishmael</cardHolderName>
      <address>Sailing Vessel Pequod, Old Harbor, Gloucester MA
01111</address>
    </exm:BookPurchase>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## Field Level Signing

The next example demonstrates field level signing, that is signing of multiple fields using the same key. This could be useful in many instances. For example, if only one or two elements within a large document contain sensitive information, there is no need to incur the overhead of signing the entire document. A more practical reason might be that multiple signatures are required for request processing, perhaps the purchasing department signs the order date, while shipping signs the shipping date.

Again, the DataPower Sign action is used but a new object, the Document Crypto Map, is added to describe the fields to be signed. This association can be seen in Figure 19-2. Notice also that the Selected Elements radio button has been selected as has the WS-Security (WSec) Method. Field level signing is available only with the WS-Security Envelope method.

**DATAPOWER X150** Help

**Configure Sign Action**

**Basic** **Advanced**

**Input**

Input:

**Options**

**Sign**

**Envelope Method**

- ☐ Enveloped Method
- ☐ Enveloping Method
- ☐ SOAPSec Method
- ☒ WSSec Method
- ☐ Advanced
- \*

**Message Type**

- ☐ SOAP Message
- ☐ SOAP With Attachments
- ☐ Raw XML Document
- ☒ Selected Elements (Field-Level)
- ☐ Advanced
- \*

**Document Crypto Map**

**Asynchronous**

☐ on ☒ off

**Key**

☒ Save

**Certificate**

☒ Save

**Output**

Output:

**Figure 19-2** Sign action with document crypto map for field level selection.

Figure 19-3 shows the details of the Document Crypto Map with two XPath statements having been added. Any number of individual XPath statements can be added (identifying single or sets of elements), allowing for many elements to be individually signed. The XPath Tool can be used to automatically generate the XPath statement from the anticipated request message.

**Configure Document Crypto Map**

Main Namespace Mappings

Document Crypto Map : SignCreditCardInfo [up]

Apply Cancel Undo Export View Log View Status Help

Admin State: ☒ enabled ☐ disabled

Comments:

Operation: Sign (WS-Security) \*

XPath Expression:   
/\*[local-name()='Envelope']/\*[local-name()='Body']/\*[local-name()='BookPurchase']/\*[local-name()='creditCardInfo']   
/\*[local-name()='Envelope']/\*[local-name()='Body']/\*[local-name()='cardHolderName']   
/\*[local-name()='Envelope']

Add XPath Tool

**Figure 19-3** Document Crypto Map.

Submitting the SOAP document previously shown in Listing 19-5 provides an example of field level signing. As per the XPath statements entered in the Document Crypto Map, the cardHolderName and creditCardInfo elements have been signed. The construction of the Signature element and its contents is similar to the document level example previously described. However, in this case we now have multiple Reference elements, each of which describes the element signed and as described previously, the digest value and its calculation methodology.

The careful observer might note the Reference URI attributes pointing to the creditCardInfo and cardHolderName elements, but you might wonder why there is an extra Reference node. In this example, a Timestamp has been added to the wsse:Security header. This allows for a mechanism to control the period in which the signature is valid. Perhaps you are authorizing a purchase, but you do not want to be locked into the content and price indefinitely. This header (created automatically by the WSSec method) ensures that an unscrupulous intermediary does not change that time period.

**Listing 19-5** Sample Field Level Signing

```

<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-WS-Security-secext-1.0.xsd" soapenv:mustUnderstand="1">
  <wsu:Timestamp wsu:Id="Timestamp-7e1b0b40-b9fd-409c-bad4-697a56c02a2f"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-WS-
Security-utility-1.0.xsd">
    <wsu:Created>2008-07-21T16:36:54Z</wsu:Created>
    <wsu:Expires>2008-07-21T16:41:54Z</wsu:Expires>
  </wsu:Timestamp>

  <wsse:BinarySecurityToken (Truncated)/>
  <SignedInfo>
    <CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="#Id-f16b389e-5193-4cbd-a5d5-92ac55c139c0">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#" />
      </Transforms>
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>uCPOLDKEDz11IwBwyXvfXgnE1cU=</DigestValue>
    </Reference>
    <Reference URI="#Id-cac033a5-42c4-4aa5-83d2-8d4424a4783b">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#" />
      </Transforms>
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>yXxcIJpMuyEwTQaoonlmtlfd27E=</DigestValue>
    </Reference>
    <Reference URI="#Timestamp-7e1b0b40-b9fd-409c-bad4-
697a56c02a2f">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#" />
      </Transforms>
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>WZzZoH+Ls6wbPKDVEkuw1idr3CQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>RlV(Truncated)</SignatureValue>
  ... (Truncated) ...
</wsse:Security>

```



Listing 19-6 shows the body of the signed SOAP envelope. Each signed element has been modified by adding a `wsu:Id` attribute that links the element to one of the Reference URI attributes contained in the `wsse:Security` header that describes the signature and generation methodology.

### Listing 19-6 Sample Field Level Signing

```
<SOAP-ENV:Body>
  <exm:BookPurchase>
    <isbn>0321228006</isbn>
    <price>19.95</price>
    <creditCardInfo wsu:Id="Id-f16b389e-5193-4cbd-a5d5-92ac55c139c0"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-WS-
Security-utility-1.0.xsd">1234567890121234-0808</creditCardInfo>
    <cardHolderName wsu:Id="Id-cac033a5-42c4-4aa5-83d2-8d4424a4783b"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-WS-
Security-utility-1.0.xsd">Ishmael</cardHolderName>
    <address>Sailing Vessel Pequod, Old Harbor, Gloucester MA
01111</address>
  </exm:BookPurchase>
</SOAP-ENV:Body>
```

## Attachment Signing

XMLDSIG allows for the signing of binary data such as images and MIME attachments, as well as XML documents. DataPower also supports this functionality. Again the signing is done via the Sign action, as can be seen in Figure 19-4. The SOAP With Attachments option is selected and the Attachment Handling option has been set to Attachments Only. It is also possible to sign both the root XML document as well as the attachment.

Options	
<b>Sign</b>	
<b>Envelope Method</b>	<input type="radio"/> Enveloped Method
	<input type="radio"/> Enveloping Method
	<input type="radio"/> SOAPSec Method
	<input checked="" type="radio"/> WSSec Method
	<input type="radio"/> Advanced
*	
<b>Message Type</b>	<input type="radio"/> SOAP Message
	<input checked="" type="radio"/> SOAP With Attachments
	<input type="radio"/> Raw XML Document
	<input type="radio"/> Selected Elements (Field-Level)
	<input type="radio"/> Advanced
*	
<b>Asynchronous</b>	<input type="radio"/> on <input checked="" type="radio"/> off
<b>Attachment Handling</b>	Attachments Only <input checked="" type="checkbox"/> Save
<b>Key</b>	BookPurchaseClient <input checked="" type="checkbox"/> Save
<b>Certificate</b>	BookPurchaseClient <input checked="" type="checkbox"/> Save
<b>Output</b>	

Figure 19-4 Attachment signing.

Listing 19-7 shows some of the interesting elements produced when signing MIME packages. Again, several fields are truncated for brevity. Of note is the Reference element that contains a URI reference to the Content-ID of the attachment. In addition the Transform element now contains an attachment canonicalization not an XML canonicalization method.

### Listing 19-7 Attachment Signing

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#" />
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1" />
    <Reference URI="cid:xml2@bookPurchaseRequest.ibm.com">
      <Transforms>
        <Transform Algorithm="http://docs.oasis-
open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-
Only-Transform" />
      </Transforms>
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>xe3pYnsXqkITHdw0UMgJLudCn7o=</DigestValue>
    </Reference>
  </SignedInfo>
<SignatureValue>(Truncated)</SignatureValue>
  <KeyInfo>
    <wsse:SecurityTokenReference xmlns="">
      <wsse:Reference URI="#SecurityToken-4ba7a5c7-5f5c-45bb-aab4-
297aceed3024" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3" />
    </wsse:SecurityTokenReference>
  </KeyInfo>
</Signature>
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <xml2@bookPurchaseRequest.ibm.com>
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <bookPurchase>
      <isbn>0321228006</isbn>
      <price>19.95</price>
      <creditCardInfo>1234567890121234-0808</creditCardInfo>
      <cardHolderName>Ishmael</cardHolderName>
      <address>Sailing Vessel Pequod, Old Harbor, Gloucester MA
01111</address>
    </bookPurchase>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## Signing Advanced Options

The advanced tab of the Sign action exposes many parameters you can customize. Figure 19-5 demonstrates a partial list, and the options depend on which signature method is being employed. WS-Security (WSS) allows for more than SOAP Security or the Raw XML signatures that are available for XML without the SOAP envelope structure. These options allow for control over the digest and signature methods, timestamps, and other DSIG characteristics.

The screenshot shows the 'Options' dialog box for the 'Sign' action. The 'Action Type' is set to 'Sign'. The 'Envelope Method' is set to 'WSSec Method'. The 'Message Type' is set to 'SOAP Message'. The 'Processing Control File' is set to 'store:///sign-wssec.xsl'. The 'Output Type' is set to 'Default'. The 'Asynchronous' checkbox is unchecked. The 'Key' is set to 'BookPurchaseClient'. The 'Certificate' is set to 'BookPurchaseClient'. The 'Signing Algorithm' is set to 'rsa'. The 'Canonicalization Algorithm' is set to 'Exclusive'. The 'Message Digest Algorithm' is set to 'sha1'. The 'SOAP Actor/Role Identifier' is empty. The 'Key/Certificate Base Name' is empty. The 'WS-Security Version' is set to '1.0'. The 'Include SOAP mustUnderstand' checkbox is unchecked.

**Figure 19-5** Signing advanced options.

## Signature Verification

Signature verification is a fairly straightforward process. The multistep policy provides a Verify action for this purpose, and it can be dropped onto the policy at any point. Conveniently a Verify action can be used to authenticate multiple (field level) signatures, document level signatures, and attachment signatures using a single methodology. As was demonstrated in the signature discussion (see “Digital Signatures on DataPower” earlier in this chapter), the public certificate

used for validation is often carried within the KeyInfo element of the DSIG message. The most typical method (we see others later in the chapter) of validating a signature is by creating a list of public certificates that match this KeyInfo value. The Validation Credential (ValCred) object is used for this purpose, and as can be seen in Figure 19-6, its use completes the Verification action.

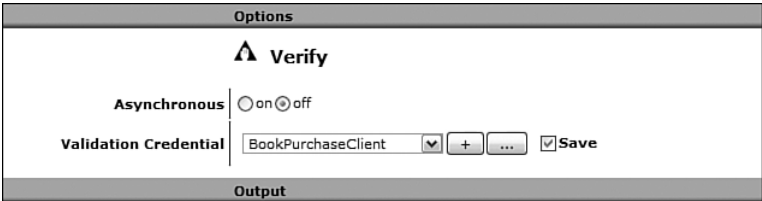


Figure 19-6 Verify action.

Figure 19-7 shows the details of the ValCred object. It contains a reference to the BookPurchaseClient Certificate object that successfully verifies signatures generated by the holder of its private key pair. In fact several certificates could have been added here to facilitate verification of additional signatures. Although this exact match mode is the most typical and efficient method available, an alternative method of verification is provided. Full certificate chain checking (PKIX) requires validation all the way up to the root certificate. This is a resource intensive operation and should only be done when a copy of the public certificate cannot be obtained and placed in the ValCred object.

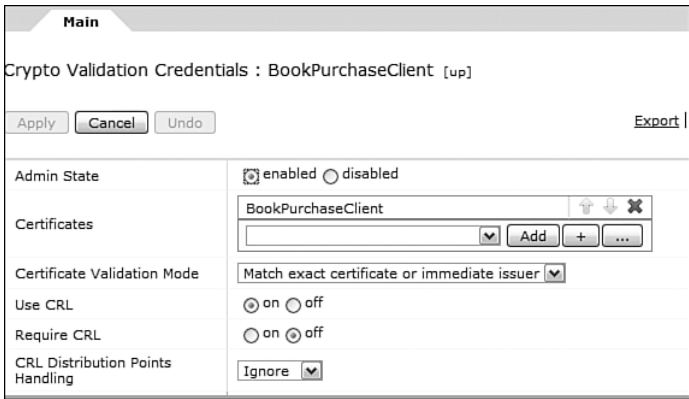


Figure 19-7 Validation credentials.

DataPower provides many optional configuration methodologies and signature verification, which in the vast majority of cases will be handled by the use of certificates, and the ValCred object is no exception. As Figure 19-8 shows, there are several parameters available from the advanced screen that may be used to fine-tune the verification process. Specifically, the ValCred can be entered in the Validation Credential field, or a variation called Signer Certificate can be used to directly identify the Certificate object. Other options may be used to turn off timestamp checking as well as restricting what algorithms are acceptable and other verification particulars.

Options

Verify

Action Type

Verify

\*

Processing Control File

store:///verify.xsl

\*

Asynchronous

on

off

Validation Credential

(none)

+

...

Save

Signer Certificate

name:BookPurchaseClient

Save

Check Timestamp Expiration

on

off

Save

Timestamp Expiration Override Period

0

sec

Save

Must check SignatureConfirmation (WS-Security 1.1)

on

off

Save

Save Verified Signatures for Later wsse11:SignatureConfirmation

on

off

Save

WS-Security 1.1: Retrieve Remote Token

on

off

Save

XPath Expressions Which Must be Signed

(empty)

Add

XPath Tool

Save

Restrict Signature Algorithm

on

off

Save

SOAP Actor/Role Identifier

Save

Add Parameter

Output

Figure 19-8 Validation advanced options.

### Digital Signature Verification in DataPower Services

Most of the DataPower services can utilize the Verify action. However, the Web Service Proxy has the additional feature of automatic document level signature verification. The public certificate must exist in a Crypto Validation Credential object. If not, the signature verification will fail with an untrusted certificate error. You can read more about WSP configuration in Chapter 10.

### Encryption and Decryption on DataPower

The previous section discussed DataPower configurations to verify the integrity of messages it receives by virtue of their digital signatures. To provide for confidentiality, DataPower provides a simple mechanism for the encryption and decryption of message data. The multistep policy editor provides Encrypt and Decrypt action icons that can be placed onto the policy in any location. Standard use cases are typically handled with little effort other than assignment of key information; although less often encountered, use cases can be managed through fine-tuned advanced parameter assignment. The following sections demonstrate the encryption of an entire document, field level encryption, and the decryption of each.

### Document Level Encryption

We'll use the sample book purchase request document presented in Listing 19-3 for the encryption and decryption examples. You might want to refer back to that listing.

The XMLENC specification allows for the encryption of individual elements or combinations of elements within the XML payload. This example will demonstrate document-level encryption using the standard XML encryption mechanism. This Envelope Method (Standard XML Encryption) does not make use of the WS-Security headers and may be used against raw XML Documents, though we are using a SOAP document in this example.

Figure 19-9 shows the completed Encrypt action. All that is required is the recipient's certificate that will be used in the encryption process. The recipient will use their private key to decrypt the message.

Input	
Input	INPUT INPUT

Options	
<b>Encrypt</b>	
Envelope Method	<input type="radio"/> WSec Encryption <input checked="" type="radio"/> Standard XML Encryption <input type="radio"/> Advanced *
Message Type	<input checked="" type="radio"/> SOAP Message <input type="radio"/> Raw XML Document <input type="radio"/> Selected Elements (Field-Level) <input type="radio"/> Advanced *
Asynchronous	<input type="radio"/> on <input checked="" type="radio"/> off
Use Dynamically Configured Recipient Certificate	<input type="radio"/> on <input checked="" type="radio"/> off <input type="checkbox"/> Save
Recipient Certificate	BookService [v] + ... <input checked="" type="checkbox"/> Save

Output	
--------	--

Figure 19-9 Document-level encryption.

Listing 19-8 shows the results of the encryption process. As we've chosen standard XML Encryption, the encryption details are contained within the body of the message (SOAP Body in this case) within a `xenc:EncryptedData` element. Recall our earlier discussion of asymmetric encryption; we discussed how the inefficiencies of asymmetric encryption are ameliorated by the creation of an ephemeral symmetric session key that is actually used in the encryption of the message data.

The session key is described in the `KeyInfo` element while the encrypted message is contained within the `xenc:CipherData`. The cipherdata is decrypted by the recipient who first decrypts the session key using their private key (the asymmetric key), identified in this case by the `dsig:KeyName` element.

Notice again that the `CipherValue` element has been truncated to condense the listing; they contain lengthy Base64 data that does not add to the discussion.

**Listing 19-8** Encrypted Document

---

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:exm="http://www.example.org/BookService/">
  <SOAP-ENV:Body>
    <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc"/>
      <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <xenc:EncryptedKey Recipient="name:BookService">
          <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
          <dsig:KeyInfo>
            <dsig:KeyName>BookService</dsig:KeyName>
          </dsig:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>(Truncated)</xenc:CipherValue>
          </xenc:CipherData>
        </xenc:EncryptedKey>
      </dsig:KeyInfo>
    <xenc:CipherData>
      <xenc:CipherValue>(Truncated)</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

---

**Field Level Encryption**

Our next example demonstrates field level encryption. This is a useful methodology as was previously discussed in the encryption introduction section. We will be encrypting the sensitive credit card information, while leaving the remainder of the message in plaintext and accessible to services that may need it for service routing or other tasks.

Figure 19-10 shows the Encrypt action required to implement field level encryption. The WS-Security (WSSec) Method has been selected, and this results in XMLENC information being placed in the WS-Security SOAP Header element. The Message Type has been set to Selected Elements. This example uses the default option of decrypting element and data, though the Encryption Type option (on advanced screen) does allow for data only encryption.

The encryption of selective elements (as with signing) requires the assignment of a Document Crypto Map. Figure 19-11 shows the details of the Document Crypto Map with two XPath statements added. Any number of individual XPath statements can be added, each indentifying a node within the document to encrypt thus allowing for many elements to be individually encrypted. The XPath Tool can be used to automatically generate the XPath statement from the sample request message.

Options

Encrypt

Envelope Method

☒ WSSec Encryption

☐ Standard XML Encryption

☐ Advanced

\*

Message Type

☐ SOAP Message

☐ Raw XML Document

☒ Selected Elements (Field-Level)

☐ Advanced

\*

Document Crypto Map

EncryptCreditCardInfo 

+

...

Asynchronous

☐ on ☒ off

Message and Attachment Handling

Message Only

Save

Use Dynamically Configured Recipient Certificate

☐ on ☒ off 

Save

One Ephemeral Key

☐ on ☐ off ☒ Save

Recipient Certificate

BookChargeService 

+

...

☒ Save

Output

Figure 19-10 Field-level encryption.

Configure Document Crypto Map

This configuration has been modified, but not yet saved.

Main

Namespace Mappings

Document Crypto Map : EncryptCreditCardInfo [up]

Apply

Cancel

Undo

Export | View Log | View Status | Help

Admin State

☒ enabled ☐ disabled

Comments

Operation

Encrypt (WS-Security) \*

XPath Expression

/\*[local-name()='Envelope']/\*[local-name()='Body']/\*[local-name()='BookPurchase']/\*[local-name()='cardHolderName']

/\*[local-name()='Envelope']/\*[local-name()='Body']/\*[local-name()='BookPurchase']/\*[local-name()='creditCardInfo']

Add

XPath Tool

\*

Figure 19-11 Document Crypto Map.

Encrypting the SOAP document in Listing 19-3 provides an example of field-level encryption. As per the XPath statements entered in the Document Crypto Map, the cardHolderName and creditCardInfo elements will be encrypted. The BookService public key will be used in the encryption of the ephemeral session key.

Listing 19-9 shows the soapenv:Header/wsse:Security contents (with truncated data) that results from the encryption process. There are many new fields added to the document that describe exactly the process that was required to encrypt the various elements.



Each field to be encrypted results in the inclusion of a `xenc:EncryptedKey` element as a child of the `wsse:Security` element. They are used to transport the session keys (and information about the asymmetric encryption applied to it) between the sender and recipient.

The `xenc:ReferenceList` element is used to contain a reference (pointer) to the data element that was encrypted using this key. And each encrypted element contains a cipher value and encryption method that can be decrypted using the `xenc:EncryptedKey` after it has been decrypted using the target's private key. Again the lengthy CipherValues have been truncated out of this display for brevity.

### Listing 19-9 Field Level Encrypted Document

```
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
1_5" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" />
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
    <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509SubjectKeyIdentifier"
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-
1.0#Base64Binary">FLKVNj2gyQw1ZdYm+rVIRYGkzFw=</wsse:KeyIdentifier>
    </wsse:SecurityTokenReference>
  </dsig:KeyInfo>
  <xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
    <xenc:CipherValue> (Truncated) </xenc:CipherValue>
  </xenc:CipherData>
  <xenc:ReferenceList>
    <xenc:DataReference URI="#G90e6b258-eD"/>
  </xenc:ReferenceList>
</xenc:EncryptedKey>
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
1_5" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" />
  <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
    <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509SubjectKeyIdentifier"
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-
1.0#Base64Binary">FLKVNj2gyQw1ZdYm+rVIRYGkzFw=</wsse:KeyIdentifier>
    </wsse:SecurityTokenReference>
  </dsig:KeyInfo>
  <xenc:CipherData xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
    <xenc:CipherValue> (Truncated) </xenc:CipherValue>
  </xenc:CipherData>
  <xenc:ReferenceList>
    <xenc:DataReference URI="#G90e6b300-11D"/>
  </xenc:ReferenceList>
</xenc:EncryptedKey>
```

Listing 19-10 shows the encrypted SOAP Body. Some of the elements are in the clear (plaintext), while those identified by the Document Crypto Map have been encrypted. Notice that the encrypted elements have been replaced by `xenc:EncryptedData` elements, and each of these has an `Id` attribute that points to a `xenc:DataReference` URI attribute thus providing the information required for decryption.

#### Listing 19-10 Field Level Encrypted SOAP Body

```
<SOAP-ENV:Body>
  <exm:BookPurchase>
    <isbn>0321228006</isbn>
    <price>19.95</price>
    <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
Id="G90e6b258-eD" Type="http://www.w3.org/2001/04/xmlenc#Element">
      <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc"/>
      <xenc:CipherData>
        <xenc:CipherValue>(Truncated)</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
    <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
Id="G90e6b300-11D" Type="http://www.w3.org/2001/04/xmlenc#Element">
      <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc"/>
      <xenc:CipherData>
        <xenc:CipherValue>(Truncated)</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
    <address>Sailing Vessel Pequod, Old Harbor, Gloucester MA
01111</address>
  </exm:BookPurchase>
</SOAP-ENV:Body>
```

Options for encryption are available on the Advanced tab of the Encrypt action. Figure 19-12 shows a partial display of the parameters and demonstrates the fine level of control available. You can, for example, select the symmetric and key transport algorithms. Referring back to Listing 19-7, we discussed the creation of individual `xenc:EncryptedKey` elements for each field that was to be encrypted. In our example there were two, but what if there were dozens, or hundreds? This could become computationally expensive to generate and then to asymmetrically encrypt the symmetric session keys.


By selecting Use Single Ephemeral Key, this process can be reduced to a single session key used for all fields. The Dynamically Configured Recipient Certificate option allows for the dynamic assignment of a public key by the storing of key information in a system variable named “`var://context/transaction/encrypting-cert`”. This could allow for the assignment via XSLT and made conditional at execution time. This advanced option may leverage unique cases where the decryption key must be dynamically assigned, again using XSLT.

Options	
<b> Encrypt</b>	
Action Type	Encrypt *
Envelope Method	<input checked="" type="radio"/> WSSec Encryption <input type="radio"/> Standard XML Encryption <input type="radio"/> Advanced *
	<input type="radio"/> SOAP Message <input type="radio"/> Raw XML Document <input checked="" type="radio"/> Selected Elements (Field-Level) <input type="radio"/> Advanced *
Message Type	
Document Crypto Map	CreditCardInfo + ...
Output Type	Default
Asynchronous	<input type="radio"/> on <input checked="" type="radio"/> off
Message and Attachment Handling	Message Only <input type="checkbox"/> Save
Use Dynamically Configured Recipient Certificate	<input type="radio"/> on <input checked="" type="radio"/> off <input type="checkbox"/> Save
One Ephemeral Key	<input type="radio"/> on <input checked="" type="radio"/> off <input checked="" type="checkbox"/> Save
Recipient Certificate	BookService  + ... <input checked="" type="checkbox"/> Save
Symmetric Encryption Algorithm	3DES-CBC <input type="checkbox"/> Save
Key Transport Algorithm	rsa-pkcs1 <input type="checkbox"/> Save
SOAP Actor/Role Identifier	<input type="text"/> <input type="checkbox"/> Save

**Figure 19-12** Advanced encryption options.

## Decryption

Decryption is a straightforward process. The Decrypt action is utilized to specify the decryption key (the private key paired to the public key used for encryption). This results in all the encrypted elements in the document being decrypted. Figure 19-13 shows the complete Decrypt action.

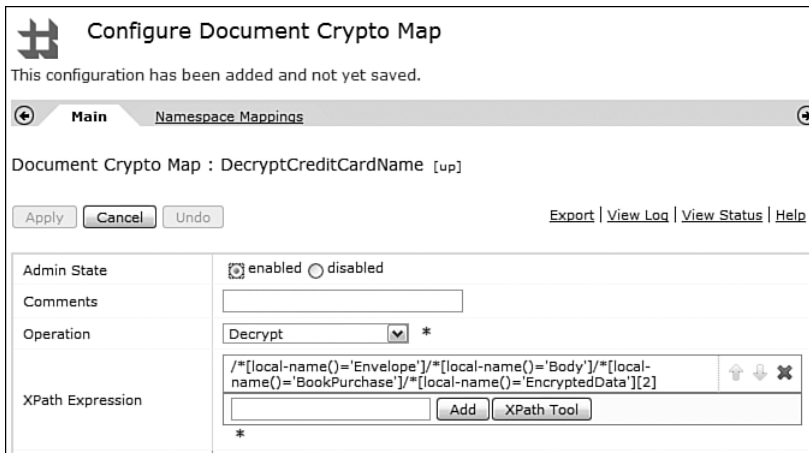
Input	
Input	<input type="text" value="INPUT"/> <input type="button" value="INPUT"/>
Options	
 <b>Decrypt</b>	
Message Type	<input checked="" type="radio"/> Entire Message/Document <input type="radio"/> Selected Elements (Field-Level) <input type="radio"/> Advanced *
Asynchronous	<input type="radio"/> on <input checked="" type="radio"/> off
Decrypt Key	<input type="text" value="BookChargeService"/> <input type="button" value="+"/> <input type="button" value="..."/> <input checked="" type="checkbox"/> Save
Output	

**Figure 19-13** Decryption action with decrypt key specified.

## Selective Decryption

It is possible to decrypt only selective fields from a document with multiple elements encrypted within it. Again a Document Crypto Map is the tool to use. However, in this case we are dealing not with the plaintext document, but rather the encrypted version. This makes the creation of XPath statements a little trickier. Had the Encryption Type (on advanced screen) been set to “Content,” the element name would not have been encrypted. Figure 19-14 shows a version of a Document Crypto Map XPath expression that selects the `cardHolderName` for decryption. However this is only possible as we know that `cardHolderName` is the second encrypted element. Another possible XPath statement would be `//*[local-name()='BookPurchase']/price/following::*[1]`; this would select the element after (following) price, in this case `creditCardInfo`. The one you use depends on your preferences of XPath notation and the dependability of the encrypted data structure.

Individual field decryption with encrypted element names is possible, though not quite as straightforward as individual field encryption. An alternative technique would be targeted encryption based on the intermediary’s public key (as used for decryption) and perhaps the use of the SOAP Actor/Role attribute. This way the intermediary can only decrypt what is intended for them without having to address the field level issues of the Document Crypto Map.



**Figure 19-14** Selective Decryption Document Crypto Map.

## Decryption in DataPower Services

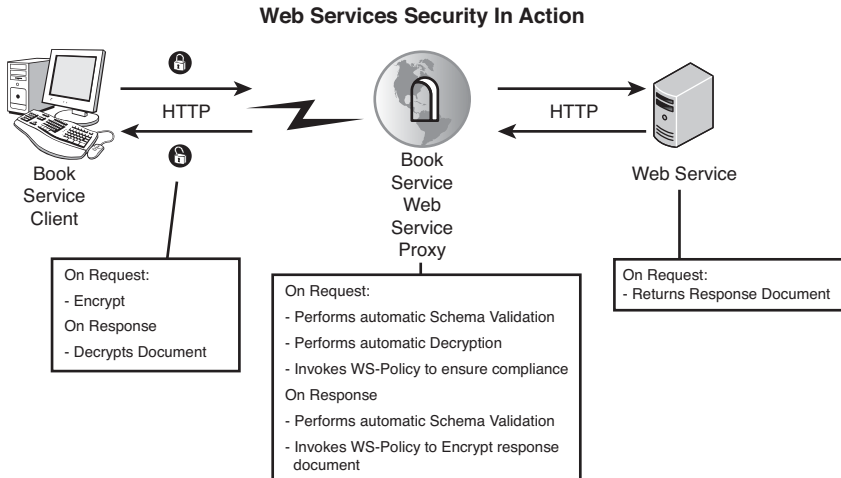
The DataPower XML Firewall, Multi-Protocol Gateway, and Web Service Proxy services can utilize the Decrypt action. However, the Web Service Proxy has the additional feature of automatic decryption when the entire message or the first child of the SOAP body is encrypted. The private certificate must exist in a Crypto Identification Credential object, or by identification via the Decrypt Key property of the WSP that is found on the Proxy Settings tab within the WebGUI.

## Putting It All Together

Now that we have presented the tools for the implementation of Web services security, we can present and implement an architecture that employs mechanisms necessary to leverage some of the techniques we have discussed.

This example demonstrates a client for a book query and purchase service that uses an XML Firewall to encrypt requests as required by a book service Web Service Proxy that uses WS-Policy for policy enforcement. This example uses the option of the client utilizing a DataPower device; however this is not a requirement. The only requirement is that the client encrypts the message using the correct keys and certificates to conform to the WS-Policy specification.

Figure 19-15 shows the architecture, and describes the configured security actions of the client and automatic processing invoked by the WSP.



**Figure 19-15** Web Services Security.

Figure 19-16 shows the multistep policy required for the book service client. An Encrypt action is used on the request rule with the BookService public certificate, and a Decrypt action is used on the response with the BookServiceClient private key.

The Web Service Proxy is capable of nearly automatic configuration by consuming a WSDL, and exposes the WSDL's service, port, and operations for fine-grained policy configuration. You can read more about WSP configuration in Chapter 10.

Having defined the WSDL and assigned the Front Side Handlers, the WSP is ready to begin receiving traffic. All the automatic features are in place. Schema validation of requests and responses, signature validation, and message decryption (if required) are all enabled.

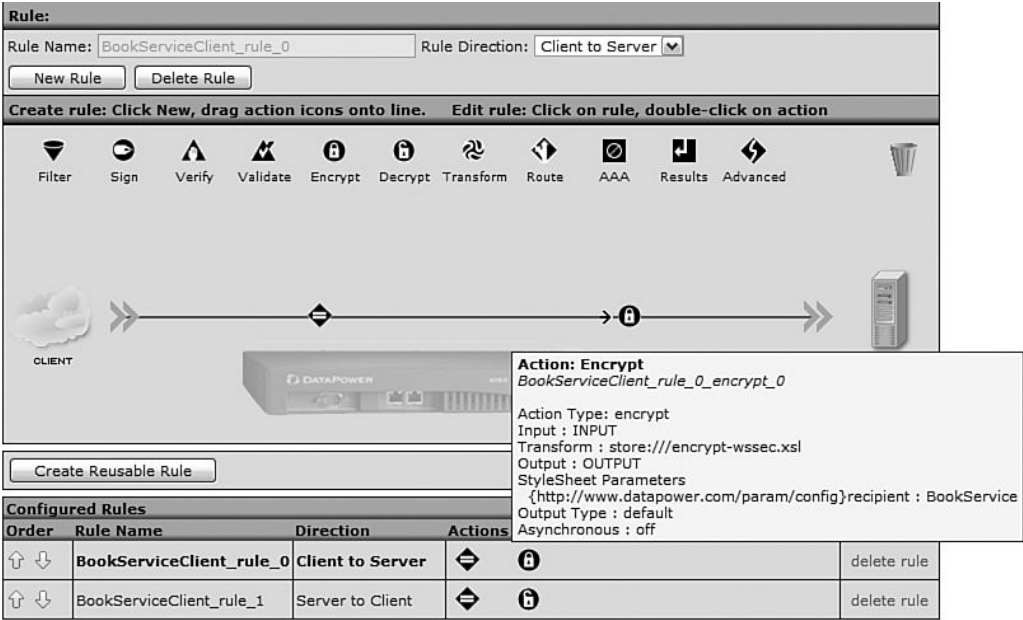
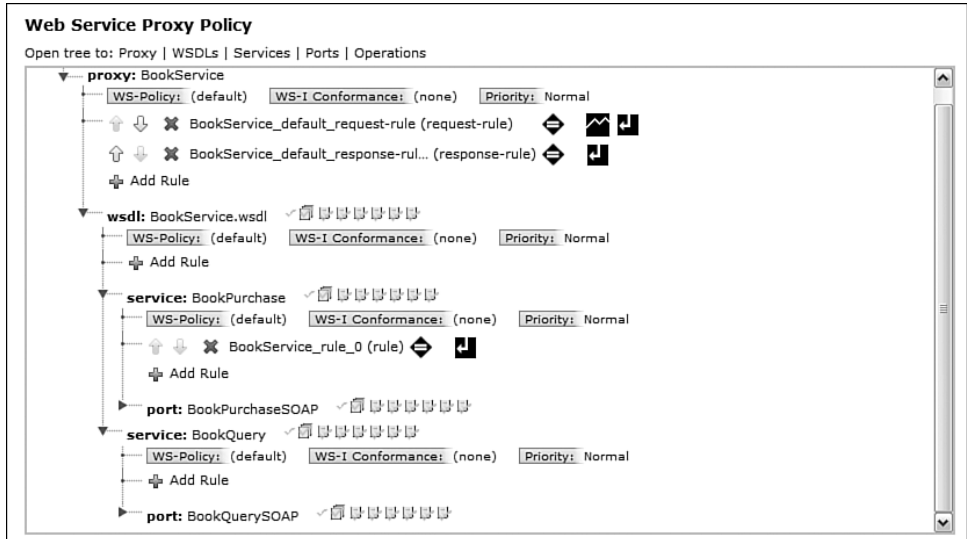


Figure 19-16 Policy with Encrypt action.

Additional processing can be assigned via the multistep policy, and Figure 19-17 shows the Policy screen. Notice that the WSDL has been exposed to allocate rules at multiple levels. Also notice that WS-Policy definitions may be assigned at various levels. As with multistep policy rules, the policy screen allows for the assignment of WS-Policy governance for the entire service, or at a more granular basis such as the operation.

In this example, notice that an external policy attachment has been defined at the port level of the BookPurchase service. External policy attachments are frequently used when the original WSDL was not authored with WS-Policy definitions. This is the WS-Policy association we desire, and we'll look at the details of its definition shortly.

We want to define a policy to enforce that messages must be encrypted. This policy will be enforced at the port level for the BookPurchase service rather than by using WSDL authoring. We will leverage the templates that DataPower provides in the store:///policies/templates/ directory. These templates are based on WS-Policy version 1.1 and 1.2, and contain a number of sample implementations such as wsp-sp-1-2-encrypted-parts-body.xml that provides the encrypt functionality we are interested in. Listing 19-11 shows the wsp-sp-1-2-sign-encrypt-order.xml template.



**Figure 19-17** Web Service Proxy.

**Listing 19-11** wsp-sp-1-2-encrypted-parts-body.xml

```
<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy"
            xmlns:sp="http://docs.oasis-open.org/ws-sx/WS-
Securitypolicy/200702">
  <dpe:summary xmlns:dpe="http://www.datapower.com/extensions">
    <dppolicy:domain xmlns:dppolicy="http://www.datapower.com/policy">
      http://docs.oasis-open.org/ws-sx/WS-Securitypolicy/200702
    </dppolicy:domain>
    <description>
      Implements WS Security Policy 1.2 - support EncryptedParts
    </description>
  </dpe:summary>
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:EncryptedParts>
        <sp:Body/>
      </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Clicking the WS-Policy button (below BookPurchaseSOAP) exposes the WS-Policy definition screen. The first step is to define a Policy Parameter Set. As we discussed earlier, the policy does not contain the implementation specific details such as keys and certificates. The Policy Parameters set allow us to assign the information necessary to perform the cryptographic operations. Figure 19-18 shows the WS-Policy screen.

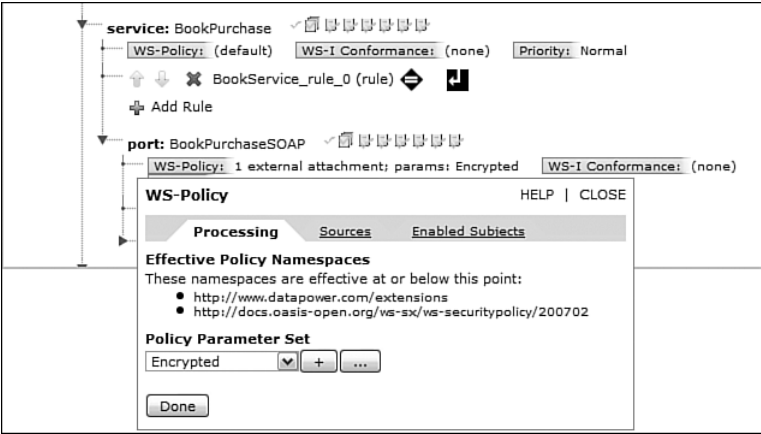


Figure 19-18 WS-Policy definition.

Figure 19-19 shows the definition of the Policy Parameters. First a Policy Domain is selected (<http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702>). Then the Assertion Filter is utilized to define parameters required for the policy. In our case, we require EncryptedParts. (You might want to refer back to the policy assertion in Listing 19-11.) Each of these filters exposes several parameters that are listed in the drop-down box below the Parameter Name column. Some of the parameters within assertions are optional, and you can refer to the Security Policy specification for definitive instruction.

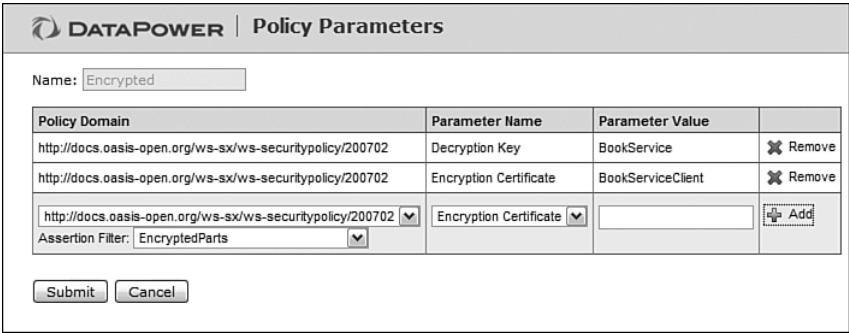
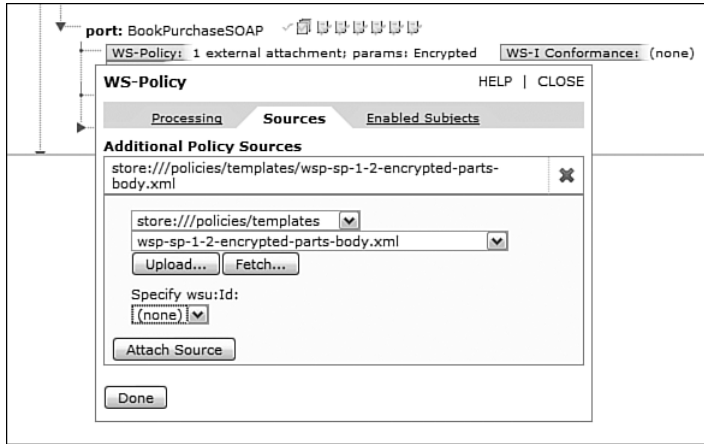


Figure 19-19 WS-Policy Policy Parameters.

Having defined the parameters necessary for the particular policy domain and assertions used, the final step in WS-Policy configuration is to define the Sources. You can do this first. In our example, the template identified previously (`wsp-sp-1-2-encrypted-parts-body.xml`) contains the policy we are interested in. Some templates contain multiple polices, and in those cases, they would be identified by their unique `wsu:Id`. Figure 19-20 shows the configured source.





**Figure 19-20** WS-Policy policy sources.

WS-Policy configuration is complete. Again, the WSP enforces the governance model that has been described. Without having to configure any multistep actions, the WSP ensures that messages are encrypted and encrypts messages on the response back to the client. Using the WSDL for policy definition affords a centralized governance policy mechanism, and any WSP service utilizing the WS-Policy conforms to the WS-Policy policies.

In addition to the WS-Policy compliance, the WSP has performed automatic schema validation. These powerful features come with minimal configuration requirements, especially when importing WSDLs complete with WS-Policy.

Listing 19-12 shows the submission of a request (again the request is in Listing 19-3) to the book service client. It encrypted the message and sent it along to the WSP. The WSP replied with an encrypted response that the book service client simply decrypted. We could have looked at the Probe and followed the encryption and decryption process through the WSP if we wanted to.

#### Listing 19-12 BookPurchase Request

```
curl --data-binary @bookPurchaseRequest.xml
http://192.168.1.35:4089/BookPurchase
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:exm="http://www.example.org/BookService/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.
org/soap/envelope/"><wsse:Security xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-WS-Security-secext-1.0.xsd"
soapenv:mustUnderstand="1"/></soapenv:Header>
<SOAP-ENV:Body>
  <exm:BookPurchaseResponse>
    <response>OK</response>
  </exm:BookPurchaseResponse>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

## Summary

This chapter has exposed some of the vulnerabilities of modern-day distributed communications. We've discussed the fundamental aspects of IT security, some of which had little to do with technology, and more to do with procedures and process management. Remember, "The best-laid plans of mice and men often go awry." And the same applies to Web Security. On a more technical note, we presented some of the differences between point-to-point security such as that found in SSL/TLS and end-to-end security found in WS-Security and supporting protocols. We dove into the details of digital signatures and encryption and their implantation on DataPower. Finally, we demonstrated a complete architecture of a Web Service Proxy utilizing WS-Policy for the enforcement of policy governance. Using these tools and techniques in a controlled and secured manner will afford you the maximum opportunities to protect organizational and client resources.