

DataPower and SSL

Cryptography is one of the most powerful security measures we have at our disposal, and DataPower has powerful cryptographic capabilities. The custom cryptographic hardware on the appliance accelerates cryptographic operations, and the ease with which complex cryptography can be utilized is astounding. This chapter explores how DataPower uses the well-known Secure Sockets Layer (SSL) to provide privacy and integrity for incoming and outgoing connections.

The Secure Sockets Layer

SSL is a standardized method of encrypting traffic on TCP/IP networks such as the Internet. The most common use is to protect Web pages. Whenever you see `https://` at the front of a URL, this means that the connection is passed over SSL and requests and responses sent through it will be encrypted. However, SSL is far more general purpose, and is commonly used for much more than encrypting Web traffic using HTTP—and indeed for more than encryption. Authentication based on cryptographic trust is a fundamental part of the SSL protocol, verifying the identity of either just the server side (the entity connected *to*) or indeed authentication of both ends of the connection (known as *mutual authentication*). SSL has been used in the wild for such obscure tasks as encrypting raw ODBC database connections when the database does not support native encryption, and providing a mutually authenticated link between servers that have no other means of authenticating each other. Indeed, general-purpose tunneling¹ software exists to create an SSL tunnel over which any TCP protocol can be passed.

¹ The term tunneling refers to the encapsulation of one protocol within another. HTTPS is in fact two protocols—HTTP is the payload protocol, and it is encapsulated or tunneled within the delivery protocol SSL. Generally, the delivery protocol is more secure than the payload protocol, and it is tunneled to provide a secure path through a less trusted network.

Cryptography

To understand SSL, it is important to have at least a basic understanding of cryptography. Cryptography can be used to solve three fundamental problems:

- **Privacy**—(Also known as confidentiality). How do I ensure that no one other than the intended parties can see what I am sending?
- **Integrity**—How can I be sure that no one has intercepted and tampered with my data?
- **Impersonation**—(Sometimes referred to as nonrepudiation). How can I be sure that I am communicating with whomever I think I am?

Each of these problems can be solved cryptographically using techniques, such as keys, certificates, signatures, and digests. We introduce these in this section.

Privacy: Algorithms and Keys, Encryption and Decryption

For the majority of people, when they think of cryptographic techniques, they think of encryption and decryption of data. In modern cryptography, this is done using cryptographic algorithms and cryptographic keys.

The cryptographic algorithm is usually published and well known. It defines a method for how to take the plain unencrypted data (usually referred to as plain text, even when it is not actually text) and turn it into encrypted data (often known as ciphertext). The key, on the other hand, must be kept secret at all costs. It is a piece of data used to alter how the algorithm behaves, in a unique way. For instance, it can work so that data encrypted with a given key and algorithm can only be decrypted by that same key and algorithm. This kind of encryption, where the same key is used to both encrypt and decrypt, is called *symmetric key encryption*.

Symmetric key encryption has one large advantage. Cryptographically speaking, it is the fastest form of encryption or decryption; the processing cost in terms of CPU and elapsed time is relatively small. However, there are also certain challenges with symmetric key encryption. Most importantly, because the same secret key is used to both encrypt and decrypt, for the encryption to be useful in communication, we need to ensure that both the sender and receiver have the same key. This might be okay for a single point-to-point connection; however, imagine trying to share many different unique keys with many different users; it is obvious that this would not scale.

Figure 18-1 shows the encryption and decryption process with a symmetric key; the same key is used to encrypt and then subsequently to decrypt the text.



Figure 18-1 Encryption using a symmetric key.

Another form of encryption uses two separate keys that have a complex mathematical relationship with each other. The relationship is such that anything encrypted with one of the keys can

be decrypted only with the other (and vice versa). Most important, there is no way² to derive one key from the other; in order to encrypt and subsequently decrypt both keys are required. This makes it perfect for exchanging encrypted messages with other people; if you have one key and another person has the related key, that person can decrypt your messages and you can decrypt his messages. Encryption using these kinds of keys is called *asymmetric key encryption*.

Keys used for asymmetric key encryption are used in a concept fundamental to SSL known as Public Key Cryptography (referred to also as Public Key Infrastructure [PKI]). One of the keys is designated as “public” meaning that it is freely shared with others, and the other key is designated as “private” and is kept secret. Because the private key cannot be derived from the public key, it is safe to share the public key with all and sundry.

Figure 18-2 shows the encryption/decryption process with a pair of keys.

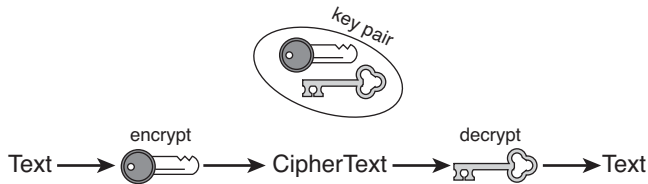


Figure 18-2 Encryption using an asymmetric key pair.

Using PKI, the problem with distribution of keys is solved. We simply share the public key and keep the private key private. Anyone who has the public key can encrypt a message and the only person who will be able to decrypt it is the holder of the private key. The only real disadvantage to using asymmetric key encryption is that it is significantly more computationally expensive than symmetric key encryption. Thus, symmetric key encryption should generally be preferred where extremely high performance is a requirement. (Although with the DataPower appliance this would have to be a *very* extreme performance requirement!)

Integrity: Cryptographic Hashes and Digests

Cryptography can also be used to solve the problem of integrity and answer the question: How can I be sure that no one has tampered with and modified my data? This is done using a different kind of cryptographic algorithm called a cryptographic hash function. The hash function takes an input and turns it into a fixed-length string called a digest (also known as a hash value).

Given the same input data, a cryptographic hash function will always produce the same digest. Moreover, the number of other input values that will produce an identical digest is extraordinarily small, and practically impossible to compute. More importantly, the chance of producing *meaningful* data that will generate an identical digest is even smaller. So, in order to guarantee the

² “No way” is of course a point in time statement. Cryptography with asymmetric keys fundamentally relies on the fact that current computing power is not enough to reasonably derive one key from the other. If someone suddenly discovers a computer that is significantly faster (by many orders of magnitude), all our existing asymmetric cryptography will be rendered insecure. With the advent of the quantum computer, this might not be as far away as you think!

integrity of a message, simply compute a digest and send it along with the message. Then, when the receiver receives the message, he can compute the digest himself and see if it matches the one you sent. If the digests do not match, clearly the message has been tampered with.

Keep in mind that cryptographic hashes are only as secure as the hash algorithms which are used to calculate the digests. For instance, a number of years ago a hash function called “MD5” was considered to be strong; it was thought that it would be extremely difficult to create a modified message that has the same digest. However, at the time of writing, it is possible to break an MD5 hash (produce a different message resulting in the same digest) in less than 20 minutes using a typical desktop computer! There are also currently concerns about the replacement for MD5, called SHA-1; we’ll see what the future holds with its superfast computers and advanced mathematics!

Impersonation: Digital Signatures

Finally, cryptography can be used to solve the fundamental problem of impersonation³ and answer the question: How do I know that the person sending me a message or otherwise communicating with me is who she says she is? This is done using digital signatures.

A digital signature is a technique that uses asymmetric key encryption and cryptographic hash functions to cryptographically “sign” a message, such that it could only have been signed by the person who holds the private key used to sign the message. This is an extremely powerful technique, because anyone who has access to the corresponding public key can verify that the signature is real.

The technique works by first calculating a digest for the message to be sent, and then encrypting that digest with the sender’s private key. The message is then sent to the receiver along with the encrypted digest. The receiver does two things. First, using the same hash algorithm as the sender, he calculates a digest of the message (without the signature); recall that the same message will always produce the same hash digest, so this should be the same as the original digest unless the message has been modified. Secondly, the receiver will decrypt the original digest using the public key of the sender; the fact that it can be decrypted by the public key means that it was encrypted with the private key. Finally, the receiver will compare the digest he calculated with the decrypted one from the message. If both digests are equal, the message has not been tampered with and only the holder of the private key could have signed it.

The flow of message digests is shown in Figure 18-3.

³ People sometimes refer to the solution of the impersonation problem as “non-repudiation.” This is a confusing and misleading term when applied to digital signatures, because it strongly implies that “someone” is behind a specific action. As will become clear, a digital signature is simply a cryptographic proof that a specific signed message was signed by a specific private key. The signature makes no claims about who currently holds that private key, or about who specifically carried out the signing. In the same way that a traditional paper signature can be repudiated by the signatory claiming that the signature is forged or was signed under coercion, a digital signature can be repudiated by the signer stating that his private key was stolen or otherwise compromised.

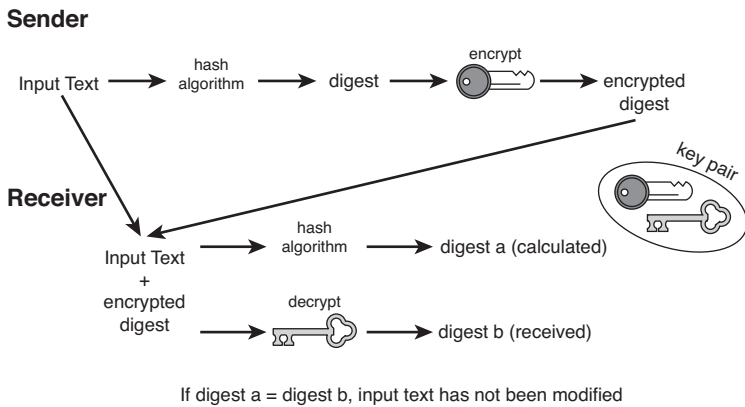


Figure 18-3 Digital signatures use hash algorithms and encryption.

Digital Certificates

The three solutions described are extremely powerful cryptographic techniques. However, none of them actually identify specific servers or people. If a message is encrypted or signed using a specific private key, you can decrypt or verify the signature using the corresponding public key. However, how does that actually help in a practical way? This is why digital certificates exist.

A digital certificate is a special way of sharing a public key with others. It contains information about the person or server to whom the certificate is assigned—perhaps a human name or a server’s hostname. It also contains a serial number and expiration date, so that it can be identified by the serial number and is valid only for a certain amount of time. Most importantly, it contains the following cryptographic information:

- A copy of the certificate holder’s public key
- A digital signature, signed by a third party (usually *not* the certificate holder)
- Information about who signed the certificate

The signer of the certificate, the trusted third party, is usually referred to as a Certificate Authority (CA). The CA is an institution that exists to sign certificates for people. It will also make *its* public key easily available, so that anyone wishing to verify that a certificate was indeed signed by that CA can do so. Note that the certificate explicitly does not include the private key, either of the certificate holder or the certificate authority—if it did, it would be useless! The private key of the certificate owner is held secretly by the owner of the certificate and is used for encryption and signing. The private key of the CA is held secretly by that CA and is used only for signing certificates. The certificate itself exists only to freely share the certificate owner’s public key with others and to “vouch for” who they are.

The role of the CA, and indeed the reason that we choose to trust them, is that it promises to verify that the person or entity requesting that a certificate be signed is indeed who they say they are. The procedure for doing so is usually based on submitting some form of proof of identity; for instance, if you want a public CA to sign a certificate for a specific domain, you need to provide evidence that you actually own that domain.

SSL and Cryptography

The SSL protocol uses all the previous cryptographic techniques to provide privacy and integrity along with optional mutual authentication. It enables privacy using both asymmetric and symmetric key encryption; an initial handshake with asymmetric keys and certificates is used to establish communication, and during that handshake, a symmetric key is shared between both sides, which is later used for all encryption. It provides for message integrity by using the shared secret key along with cryptographic hash functions to ensure that the content of messages sent along an SSL connection is not altered. Finally, it provides mutual authentication by using signed digital certificates to authenticate both sides to each other during the handshake.

Client and Server Roles

As with most TCP protocols, in an SSL conversation, the side that initiates the connection is referred to as the Client, and the side that accepts the connection is called the Server. In the Web browser case, it is obvious which is the Client (the Web browser) and which is the Server (the Web server). However SSL, and in particular HTTP, is often used in server-to-server communications, and in those instances, it is perfectly possible for two “server class” nodes to play either the Client or the Server role.

SSL Authentication and Mutual Authentication

The SSL protocol provides for one or both sides of a conversation to authenticate to each other. The entity in the role of the Server must always authenticate itself; that is, as part of the negotiation for an SSL connection, it always presents a signed certificate as its identity. The Client can also *optionally* authenticate itself with a signed certificate, but this is not a mandatory part of the protocol. When both sides authenticate, the connection is said to be “mutually authenticated.”

The SSL Handshake

Enough theory! Let’s take a scenario where two DataPower appliances need to communicate with each other, where they need to authenticate each other, be certain that data is not intercepted, and be sure that the data they are sending and receiving has not been modified in transit. Let’s explore how the concepts described previously are used in SSL.

The scenario we use is comprised of two DataPower appliances, one deployed in the Demilitarized Zone (DMZ) as a security gateway, and the other deployed in a backend server zone performing the function of an Enterprise Service Bus (ESB). Let’s call them DMZDP and ESBDP. The connection between the two needs to be encrypted (to ensure that, if for some reason network security is compromised, data cannot be simply sniffed over the wire by a third party) and mutually authenticated (such that DMZDP can be sure that it is actually connecting to ESBDP, and also that ESBDP can be sure that only DMZDP can connect to it). The scenario is depicted in Figure 18-4.

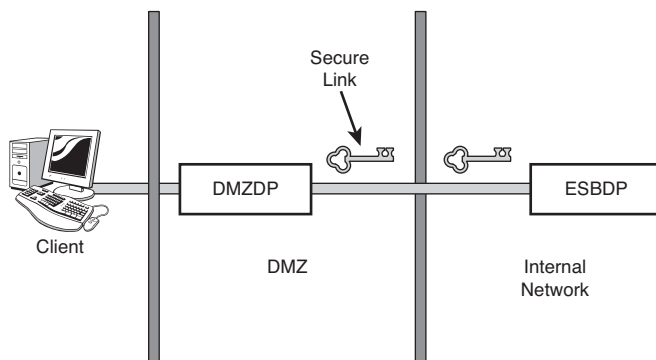


Figure 18-4 Communication between the DMZ and the ESB on the internal network.

This section presents a simplified explanation of the SSL handshake between the two devices, covering the points that are directly relevant to the usage inside DataPower. Significantly more detail can be found in RFC2246 and from many other online sources if required.

The Hello Message

Let's say that DMZDP initiates the SSL connection to ESBDP, thus making DMZDP the "client" and ESBDP the "server." DMZDP opens up a TCP socket, and when that is connected, it sends an SSL "hello" message. Inside that hello message is a list of ciphers (hash functions and encryption algorithms) that DMZDP is capable of using. These are configurable on DataPower as shown later in the chapter.

ESBDP receives the "hello" message and chooses which of the ciphers it wants to use from the ones that DMZDP has sent. If you configure two devices to communicate using SSL, both of them must be configured so that they have at least one set of ciphers (hash functions and encryption algorithms) in common! ESBDP sends back its own "hello" message, which contains the cipher that it has chosen to use for this SSL session.

ESBDP's "hello" message also includes a signed digital certificate. This is the certificate that ESBDP uses to identify itself. The digital certificate contains a public key, and ESBDP holds the corresponding private key in secure storage on the appliance. Who is the certificate signed by? Well, a trusted third party, of course!

DMZDP receives the "hello" message from ESBDP and looks at the certificate. The first thing it does is check whether the certificate is expired. If it is, DMZDP refuses to communicate and closes the connection. Next, DMZDP will validate the signature of the certificate. This means that DMZDP must know where to find the public key of the trusted third party that signed it. To trust ESBDP's certificate, DMZDP must have been preconfigured either with the public key of the signer of ESBDP's certificate or with a copy of the certificate itself. After the signature on the certificate has been validated, DMZDP retrieves ESBDP's public key from the certificate.

This interaction is shown more clearly in Figure 18-5.

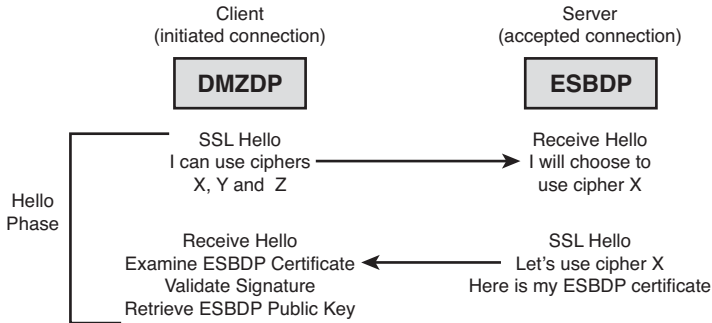


Figure 18-5 The hello phase.

Key Exchange

Recall that asymmetric key encryption is relatively expensive, and should be used sparingly. The SSL protocol recognizes this, and only uses it for the initial handshake; all further communication is done using symmetric key encryption. Thus the next stage for DMZDP is to generate a secret key that is used for symmetric key encryption. To send this secret key securely to ESBDP, DMZDP encrypts it using the public key retrieved from ESBDP's certificate, ensuring that only the holder of the corresponding private key (ESBDP) can decrypt it. Finally, because we are using mutual authentication here, DMZDP sends its own certificate (also signed by a trusted third party) to ESBDP.

ESBDP retrieves the message containing the secret key and DMZDP's certificate, and starts by examining the certificate. First, it checks to see whether the certificate is expired; if it is, ESBDP refuses to communicate further and terminates the connection. Next, ESBDP validates the signer of the certificate—clearly ESBDP must have been preconfigured with the public key of the signer of DMZDP's certificate or the certificate itself.

After the signature on DMZDP's certificate has been validated, ESBDP retrieves the secret key that was sent along with the message and decrypts it using its private key. It then sends a "finish handshake" message back to DMZDP—but this time the message is encrypted using the secret key.

DMZDP receives the "finish handshake" message and decrypts it using the secret key. The fact that it is able to decrypt it means that ESBDP must have held the private key corresponding to the public key of the certificate it had presented earlier. Both sides have now authenticated each other, and both sides now have access to a shared secret key. They can now communicate with each other using the shared secret key for symmetric key encryption; this key is used for the rest of the SSL session.

The full SSL handshake as described is depicted in Figure 18-6.

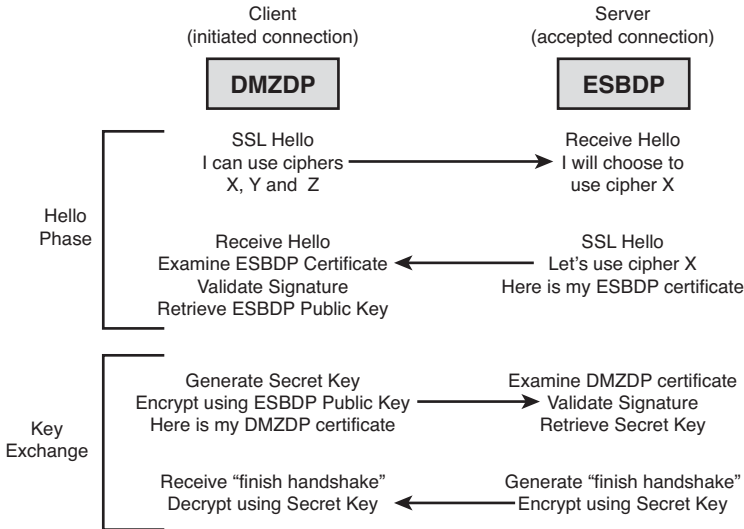


Figure 18-6 The SSL handshake.

Whew—I bet you never guessed so much was happening behind the scenes when you enter `https://` in your browser! It seems complicated, and because of that many people believe that SSL is a drag on performance. That certainly used to be the case, but with the speed of today's modern networks and computers, if things are configured properly you should not fear SSL (and certainly not forego it) for performance reasons.

SSL ID Caching

One extremely important aspect of the previously described handshake is that if it can be helped, we do not want to repeat it. It is a relatively expensive operation. The SSL protocol allows for caching of the secret keys, which is a configurable option. If caching is enabled, as it is by default, when DMZDP wants to initiate a new SSL connection to ESBDP, rather than renegotiating and agreeing a new session key, they can simply agree to use the already existing shared secret key. Because they previously negotiated the key, and they are the only two nodes that have access to it, it is a fair assumption that reusing the same key will be safe. It also means that they do not have to go through all those computationally expensive asymmetric key operations. This directly affects performance of SSL.

By implication, this means that when load balancing SSL over a number of servers, it is extremely important to send a client back to the same server if at all possible, because that server will have the cached secret session key and will be able to reuse it.

TIP — LOAD BALANCING SSL

If SSL connections are load balanced, the load balancer should always endeavor to maintain affinity, otherwise performance will be directly impacted.

Configuring SSL in DataPower

SSL configuration in DataPower often appears complicated at first glance. It requires a number of configuration objects to be created, and the relationship between these objects is sometimes hard to understand. We will examine and define those objects, show where they are used and what the relationships are between them, and show how they can be easily attached to services to allow for strongly defined incoming and dynamic outgoing SSL configurations.

Configuration Objects

This section defines the configuration objects specific to SSL, and shows where they are configured and how they are used. The cryptographic configuration objects can be accessed using the left menu bar as with all objects, but there is also a dedicated Keys and Certificates Management button on the Control Panel, which leads to the page shown in Figure 18-7.



Figure 18-7 The keys and certificates management page.

Figure 18-8 contains a diagram of the relationship between the objects required when configuring the use of SSL in DataPower.

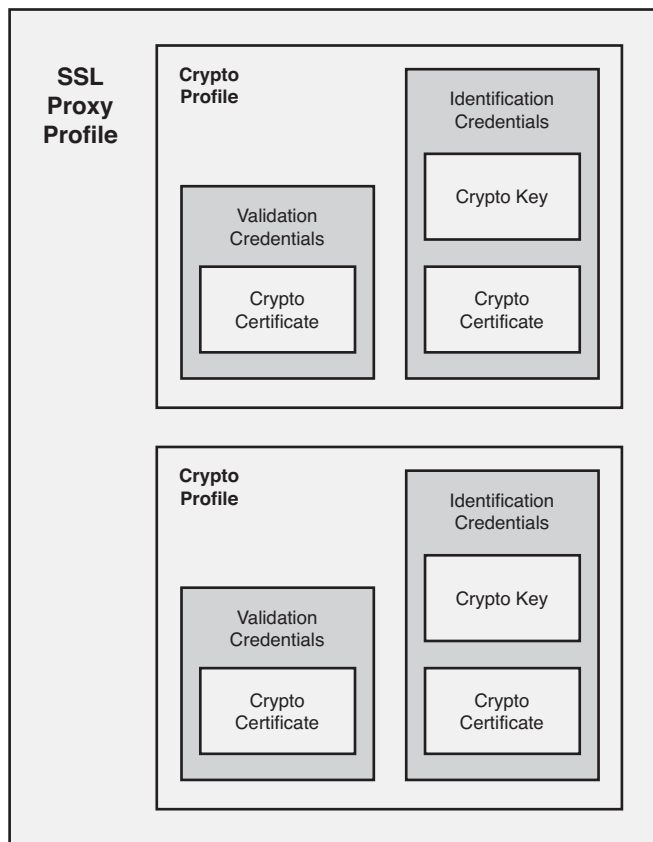
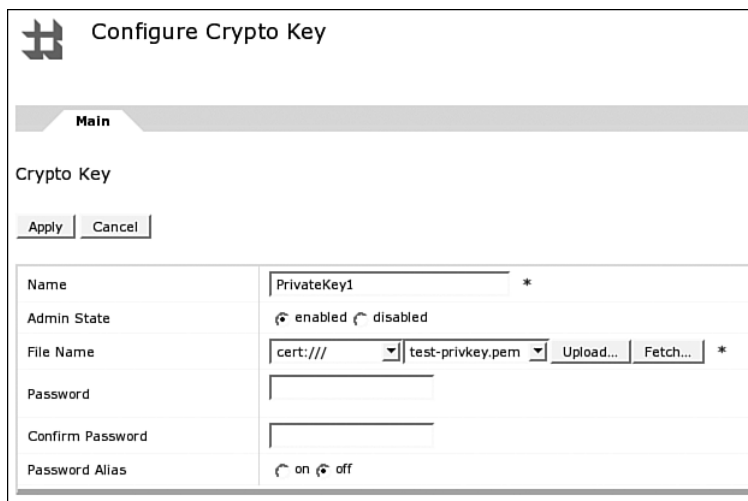


Figure 18-8 The relationships between the configuration objects.

Each of these objects is explained in turn.

The Crypto Key Object—Private Keys

The simplest DataPower object used for SSL communication is the Crypto Key object, shown in Figure 18-9. The Crypto Key object is an abstraction that contains a reference to a file stored on the DataPower directory that contains a private key. The file is stored in the secure cert: directory, and after it is uploaded to the device, it can only be used by the device itself; it cannot be retrieved or copied off the device in any way, including for backup purposes. A Crypto Key object can contain a password, which is used by the system to access the key. This is sometimes required by local security policies, which do not take into account the fact that the key is already stored on the appliance's secure file system and cannot be copied off the device.



Configure Crypto Key

Main

Crypto Key

Apply Cancel

Name	PrivateKey1 *
Admin State	<input checked="" type="radio"/> enabled <input type="radio"/> disabled
File Name	cert:///test-privkey.pem Upload... Fetch... *
Password	<input type="password"/>
Confirm Password	<input type="password"/>
Password Alias	<input checked="" type="radio"/> on <input type="radio"/> off

Figure 18-9 A Crypto Key object.

If a password is to be used, there are two options; it can either be a plaintext password, which will be stored in the configuration file of the device, or it can be a password alias, which then references a triple-DES encrypted password stored using the CLI password-map command (more information on this is in the published documentation). We recommend that, if passwords are required, password aliases referencing encrypted password maps should be used, because a plaintext password might be visible in an exported configuration file.

Note that the Crypto Key object name is merely a reference, and does not have to match or in any way relate to the key filename. Abstracting in this manner stands you in good stead for promotion of configuration through different environments, as discussed in Chapter 15, “Build and Deploy Techniques.”

The Crypto Certificate Object—Digital Certificates

Just as a Crypto Key object is a reference to a file containing a private key, similarly the Crypto Certificate object is a reference to a file stored on the DataPower directory that contains a digital certificate. A Crypto Certificate object is shown in Figure 18-10.

Again, if required by local policy, a Crypto Certificate can be protected using a plaintext password or a password alias; the password alias should be used in preference. The function to use a password is usually used only when both the certificate and the respective private key are stored in the same file. A Crypto Certificate object also has an additional configuration parameter, called Ignore Expiration Dates. As it sounds, this parameter allows usage of certificates that are out of date.

Figure 18-10 A Crypto Certificate object.

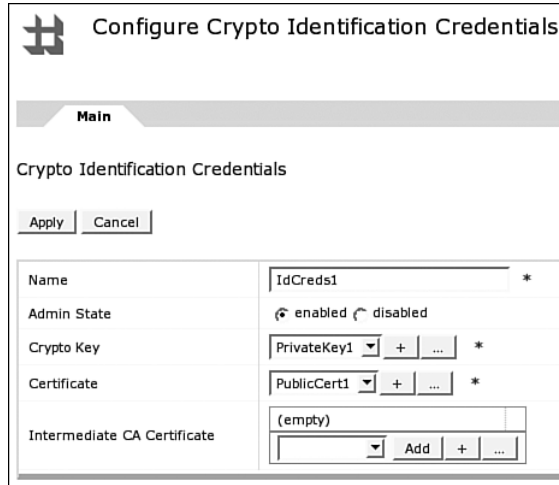
By default, the Ignore Expiration Date parameter is disabled. In this default state, that means that any object which references this Crypto Certificate object will be marked as down and unusable if the Crypto Certificate object becomes marked down because the certificate it represents is out of date. If you choose to set Ignore Expiration Date to enabled, this will not happen; certificates will be used irrespective of the dates in the certificate, and objects which reference the Crypto Certificate object will remain available for use, which might have unexpected consequences. Of course, this is a bad idea! Making cryptographic certificates be accepted after their expiration date increases the potential attack surface; the time limitation and expiration is a security feature and should be treated as such.

Certificates exist in the wild in a number of file formats, including PEM, DER, CER, PFX, and P12. If a certificate is expected to be run on DataPower but is not in a format that DataPower understands, the external open source tool OpenSSL can be used to convert between the formats to arrive at an acceptable format for DataPower.

The Crypto Key and Crypto Certificate objects can be used for more than SSL authentication; they are a general abstraction that can also be used for encryption and decryption of documents in addition to performing digital signatures. The SSL use case is only one of many that use this configuration object; for other examples, see Chapter 19, “Web Services Security.”

Crypto Identification Credentials—Who Am I?

The Crypto Identification Credentials object is a configuration object that represents an “identity” for a service on the DataPower appliance. It consists of a Crypto Key object and a Crypto Certificate object; that is, a public/private key pair. A Crypto Identification Credentials object is shown in Figure 18-11.



Name	IdCreds1 *
Admin State	<input checked="" type="radio"/> enabled <input type="radio"/> disabled
Crypto Key	PrivateKey1 + ... *
Certificate	PublicCert1 + ... *
Intermediate CA Certificate	(empty) [dropdown] Add + ...

Figure 18-11 A Crypto Identification Credentials object.

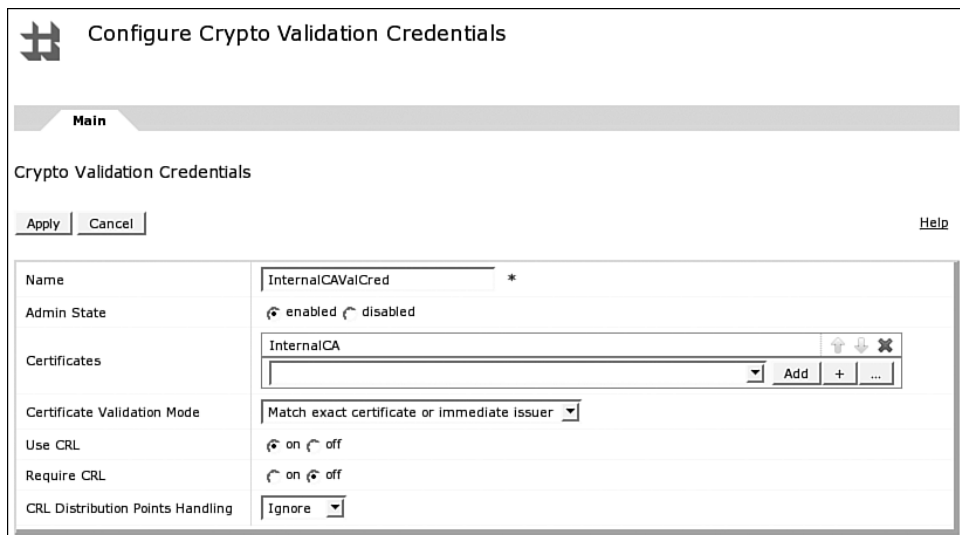
A Crypto Identification Credentials object can include an intermediate CA certificate. This is the certificate that has been used to sign the server’s certificate, but it, in turn, is signed by a trusted third party; this is known as a trust chain. If this intermediate certificate is provided, it will be sent along with the server certificate as part of the SSL handshake, in order for the client to validate the intermediate certificate while validating the actual server certificate.

The identification credentials can be used both when acting as a server, where the Crypto Certificate object contains a pointer to the certificate that is presented as the server certificate, and when acting as an SSL client, where the Crypto Certificate object contains a pointer to the certificate to be used as the client certificate. In a configuration where DataPower is acting solely as an SSL client and mutual authentication is not required, no identification credentials object is required.

Crypto Validation Credentials—Whom Do I trust?

A Crypto Validation Credentials object, as shown in Figure 18-12, consists of a list of Crypto Certificate objects. These are the “trusted third party” certificate objects we choose to use when deciding whether to trust a signed certificate.

Validation Credentials are often known as the “trust store” in other products. They can contain the public certificates of signers, which are used to validate, or they can contain copies of the certificates themselves (of course without the corresponding private keys).



Configure Crypto Validation Credentials

Main

Crypto Validation Credentials

Apply Cancel Help

Name	InternalCAValCred *
Admin State	<input checked="" type="radio"/> enabled <input type="radio"/> disabled
Certificates	InternalCA <input type="text"/> Add + ...
Certificate Validation Mode	Match exact certificate or immediate issuer ▼
Use CRL	<input checked="" type="radio"/> on <input type="radio"/> off
Require CRL	<input type="radio"/> on <input checked="" type="radio"/> off
CRL Distribution Points Handling	Ignore ▼

Figure 18-12 A Crypto Validation Credentials object.

TIP—POPULATING THE VALIDATION CREDENTIALS

One of the options to successfully trust an incoming client certificate is to have a copy of that certificate itself in your validation credentials (rather than the certificate of the signer, which would be more common). However in order to connect to an arbitrary SSL service, it might not always be easy to get hold of the certificate so that we can import it into DataPower.

Well, remember that as part of the SSL handshake, the server actually sends over a copy of its signed certificate. Thus all you need to do is to initiate an SSL handshake with a suitable tool, and keep hold of the certificate once it has been sent over the wire. The easiest tool to use for this job is your desktop Web browser—simply accept the certificate into your browser's trust store, and then export it from there for an instant copy!

In fact, because the SSL handshake is complete before any application data is ever sent over the wire, this trick can be used to retrieve the public certificate for any SSL enabled service, not just HTTPS. This is extremely useful if, say, you need to create a validation credential to connect to say an LDAP server over SSL.

The Crypto Validation Credentials object can be set to automatically contain all the certificates in the pubcert: directory on the appliance by clicking the Create Validation Credential from pubcert button. Alternatively, to not trust all the well-known signers, or to only trust specific internal ones, individual certificate objects can be added to the list. This list is the definitive list of trusted signers for any SSL configuration that uses this object.

TIP—WHOM SHOULD I TRUST?

By including a signer certificate in a validation credential, you are making an explicit statement saying that “I will implicitly trust any certificate signed by this certificate authority.” This might not always be what you want!

For example if you include a well-known signer inside a validation credential, and use mutually authenticated SSL, the SSL handshake will successfully complete with any certificate signed by that well-known signer. This includes the ubiquitous J.R. Hacker who applied last week with some photocopied paperwork to prove his identity.

There are two special options for Crypto Validation Credentials objects that define how the certificates are used to validate signatures of peer certificates. The first option, the default, to match the exact certificate or immediate issuer, does exactly that—it only validates the signature if the certificate presented exactly matches one of the certificates in the list of validation credentials, or is signed directly by one of them. The second option performs full chain checking so that any issuer certificates presented must also match; this generally isn’t necessary when dealing with trusted business partners who have shared their certificates with you, but might be needed when accepting connections from the general public.

CERTIFICATE DIRECTORIES

There are three directories on the DataPower appliance that are commonly used to store certificates. These are the `cert:`, `sharedcert:`, and `pubcert:` directories. Although all three are identical, and in theory certificates can be stored in any of them, each exists to maintain a convention, and following the convention will help avoid confusion.

- **Cert**—This directory should contain certificates that have been uploaded (or generated on the appliance) and are destined to be used for a particular application. These are not shared among domains.
- **Pubcert**—This directory includes the common root certificate authorities that would usually be found in most common Web browsers.
- **Sharedcert**—This directory should contain certificates that have been uploaded to or generated on the device, which are intended to be shared across all domains. This might be a good place to store an internal root certificate authority’s certificate.

Although these are indeed only a convention, mixing the certificates by putting them in the wrong directory can really confuse matters and should be avoided. Finally, note that certificates, keys, or any other such sensitive material should not be stored on the `local:` directory. The certificate directories are there for a reason—use them!

Crypto Profiles—Wrapping It all Together

A Crypto Profile puts together a set of configuration objects used in SSL communications. This is the basic unit of SSL configuration in DataPower. A Crypto Profile object is shown in Figure 18-13.

The screenshot shows the 'Configure Crypto Profile' window. It has a title bar with a DataPower icon and the text 'Configure Crypto Profile'. Below the title bar is a 'Main' tab. The main area is titled 'Crypto Profile'. There are 'Apply' and 'Cancel' buttons. Below these are several configuration fields:

Name	MyCryptoProfile *
Admin State	<input checked="" type="radio"/> enabled <input type="radio"/> disabled
Identification Credentials	IdCreds1 ▾ + ...
Validation Credentials	InternalCAValCred ▾ + ...
Ciphers	DEFAULT
Options	<input checked="" type="checkbox"/> Enable default settings <input checked="" type="checkbox"/> Disable SSL version 2 <input type="checkbox"/> Disable SSL version 3 <input type="checkbox"/> Disable TLS version 1 *
Send Client CA List	<input checked="" type="radio"/> on <input type="radio"/> off

Figure 18-13 A Crypto Profile object.

A Crypto Profile contains a crypto identification credential if one is used in this Crypto Profile (for instance if acting as an SSL server, or if we are an SSL client and we want to use client authentication). It also contains a crypto validation credential if one is used in this Crypto Profile (for instance if acting as an SSL client, or if acting as an SSL server and wanting to accept client authentication).

In addition, a Crypto Profile contains a number of options for deciding which algorithms and which version of the SSL protocol to use. These options are detailed in the documentation; in all likelihood the defaults will suffice, but if you have specific requirements these can be configured.

The SSL Proxy Profile—Using SSL for Proxy Communication

The SSL Proxy Profile is a further level of abstraction for configuring some services on DataPower to use SSL. An example of an SSL Proxy Profile is shown in Figure 18-14.

SSL Proxy Profiles can be either “forward,” meaning that they are used when DataPower is an SSL client, “reverse,” used for DataPower as an SSL server, or “two-way,” where the proxy will be acting as both client and server (usually this means client to appliance and appliance to back end server). The direction here is simply a description of the deployment pattern; that is, how is SSL going to be used.

Configure SSL Proxy Profile

Main

SSL Proxy Profile

Apply Cancel

Name	AppServerProfile *
Admin State	<input checked="" type="radio"/> enabled <input type="radio"/> disabled
SSL Direction	Reverse *
Reverse (Server) Crypto Profile	MyCryptoProfile + ... *
Server-side Session Caching	<input checked="" type="radio"/> on <input type="radio"/> off
Server-side Session Cache Timeout	300 seconds
Server-side Session Cache Size	20 entries (x 1024)
Client Authentication Is Optional	<input checked="" type="radio"/> on <input type="radio"/> off

Figure 18-14 An SSL Proxy Profile object.

- When DataPower acts as an SSL client, a validation credential is usually needed (if no validation credential is present we will validate using the certificates in the pubcert: directory) and an identification credential is required if mutual authentication is in use. The SSL direction would be set to Forward, and a “forward” Crypto Profile would be defined.
- When DataPower acts as an SSL server, an identification credential is required, and also a validation credential if mutual authentication is in use. The SSL direction would be set to Reverse, and a “reverse” Crypto Profile would be defined.
- When DataPower is acting as both an SSL server (receiving an SSL connection from an external client) and an SSL client (connecting over SSL to a back end server), *two* sets of identification and validation credentials are required, one for each connection. The SSL direction would be set to “two-way” and *both* a “reverse” and a “forward” Crypto Profile would be defined.

This is really the *raison d’être* of the SSL Proxy profile. The SSL Proxy Profile gives us a way of grouping the required validation credential and identification credential objects that are relevant to the required deployment pattern. A Crypto Profile is assigned to the “Forward Crypto Profile,” the “Reverse Crypto Profile,” or both, depending on the intended usage. These Crypto Profiles contain the entire cryptographic configuration needed by the SSL Proxy Profile.

Settings specific to the usage of caching of SSL in this configuration are also set here. These include server and client side SSL caching, which is on by default, and the cache sizes and timeouts for both server and client side.

Finally, the SSL Proxy Profile has a configuration option called “Client Authentication Is Optional.” Client authentication is the component of the mutually authenticated handshake where

the client authenticates itself to the server. By default, this option is “Off”—meaning that by default, client authentication is mandatory if a validation credential has been configured. However, authenticating the client is not a mandatory part of the SSL specification. Most Web traffic (requests for HTML pages) that use SSL usually do not enforce client authentication; they mainly require encryption, and use other mechanisms to authenticate their clients. For Web services traffic, there are also valid use cases where encryption and authentication of the server certificate are enough, and, therefore, we do not need to use client authentication. In that case, this option may be disabled.

Creating Targeted Crypto Profiles

Crypto Profiles can be created using the Crypto Profile object page. However, there is also a useful wizard that is accessible anywhere that a Crypto Profile might be required, and this wizard provides a single page that allows you to configure everything the Crypto Profile needs.

The wizard, which in this instance is accessed by clicking the + sign next to the SSL Server Crypto Profile in an XML firewall configuration, is shown in Figure 18-15.

Configure Server Side SSL Help

General **Advanced**

Apply Cancel

Profile Name: MyServerCryptoProfile *

Server Credentials The server certificate presented to clients making SSL connections to DataPower appliance.

Server Private Key: cert:/// test-privatekey.pem Upload... Fetch...

Private Key Password: Confirm Password

Use Password Alias: ☐ on ☒ off

Server Certificate: cert:/// test-accert.pem Details... Upload... Fetch...

Server Certificate Password: Confirm Password

Use Password Alias: ☐ on ☒ off

Trusted Clients Trusted certificate(s) presented by clients to the DataPower appliance when using two-way SSL (mutual authentication).

Authenticate/Validate Certificates: ☒ on ☐ off *

Trusted Certificates/Certificate Authorities: outcert:///First-Data-Digital-Certificates-CA.pem Delete

pubcert:/// {name} Add Details... Upload... Fetch...

Password: Confirm Password

Certificate Validation Mode: Match exact certificate or immediate iss

Send CA List To Client: ☐ on ☒ off

SSL Options SSL handshake behavior options.

Options: ☒ Enable default settings ☒ Disable SSL version 2 ☐ Disable SSL version 3 ☐ Disable TLS version 1 *

Figure 18-15 The Crypto Profile creation wizard for a Server Crypto Profile.

The wizard creates an identity credential (here referred to as a “server credential”) and a validation credential (here referred to as “trusted clients”). It also allows setting of the relevant options regarding version, and on the Advanced tab the ciphers to use, and so on.

SSL Usage Pattern in DataPower

The most common usage pattern of the DataPower appliance is that of a proxy. That is, it accepts a connection over the network, adds value in the form of transformation, AAA, routing, and so on, and then initiates a separate new connection to a backend server to send it the request and receive the response, which it then sends to the client over the original connection on that side. It is, therefore, possible to use SSL on both of these connections—the inbound and the outbound.

Due to the nature of SSL, it is impossible to simply pass the connection on to the backend “transparently.” It is impossible for one node to pretend to be another, because they do not possess the private key associated with that other node’s certificate. This means that, in the proxy pattern, there is not and cannot be any direct relationship between the incoming and outgoing SSL connections; they are two separated and isolated connections. Of course, it is possible to explicitly share the private keys between the backend and DataPower, so that DataPower can act as if it were the backend server; indeed the two-way Crypto Profile makes this configuration easy to implement. However in most cases there would be no need to share. DataPower would simply *become* the main SSL endpoint, and it would have the only copy of the private key.

Of course, there is no absolute requirement to use SSL both on the inbound and outbound configuration. There may be valid use cases for a service to use SSL on its inbound connection, but to make a plain unencrypted connection to the backend. (Of course this assumes that you trust everyone on your internal network, which is likely a bad idea; a large percentage of real-world attacks are internal, and internal networks are almost never secured as well as they should be, not to mention that there may be privacy laws that you would have to ensure that you respect.) More rarely there might be a use case for going the other way—taking an unencrypted connection inbound, and making an outbound connection with SSL.

Whichever pattern is used, the SSL configuration as defined previously must be attached to a service for inbound connections, or configured for use as an outbound connection. Doing so is a simple matter—a relief after the somewhat complex procedure for configuring the abstractions in the first place! It is when you see how easy it is to attach the configurations that the power of the abstraction becomes obvious.

Using SSL—Inbound Configuration

After an SSL configuration has been created, it is easy to add that configuration to a service. For instance, a specific Crypto Profile can be reused among multiple XML Firewalls within the same domain, and as part of one or more SSL Proxy Profiles. The same SSL Proxy Profile can be added to multiple services of different types, such as HTTPS or FTP Server FSHs for attaching to Web Service Proxies or Multi-Protocol Gateways, or any other type of service requiring SSL.

To attach an SSL Crypto Profile to an XML Firewall object, simply select it as the SSL Server Crypto Profile when defining the firewall, as shown in Figure 18-16.

Front End

Device Address

0.0.0.0

Select Alias

*

Device Port

16384

*

SSL Server Crypto Profile

MyServerCryptoProfile

+

...

Request Type

SOAP


Request Attachments

Strip

Figure 18-16 SSL Server Crypto Profile configured on an XML Firewall service.

Keep in mind that when you update an XML Firewall to use HTTPS, the HTTP listener will no longer be available, so all your clients will need to change their URLs to use HTTPS.

The same Crypto Profile might be part of an SSL Proxy Profile, as shown in Figure 18-17, which can then be attached to an HTTPS FSH object or an FTP Server FSH object or anywhere else an SSL Proxy Profile can be used.

 Configure SSL Proxy Profile

Main

SSL Proxy Profile : AppServerProfile [up]

Apply

Cancel

Undo

Admin State	<input checked="" type="radio"/> enabled <input type="radio"/> disabled
SSL Direction	Reverse * <div></div>
Reverse (Server) Crypto Profile	MyServerCryptoProfile + ... *
Server-side Session Caching	<input checked="" type="radio"/> on <input type="radio"/> off
Server-side Session Cache Timeout	300 seconds
Server-side Session Cache Size	20 entries (x 1024)
Client Authentication Is Optional	<input type="radio"/> on <input checked="" type="radio"/> off

Figure 18-17 The same Crypto Profile configured on an SSL Proxy Profile.

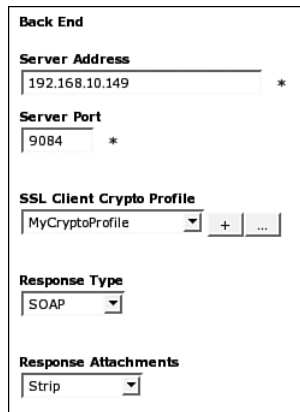
The FTP Server FSH usage of the proxy profile is more complicated, as described later in this chapter, but the fundamental principle holds true: Assigning an SSL configuration (in this case in the form of an SSL Proxy Profile) to a service is trivial, after all the cryptographic configuration is complete.

Using SSL—Outbound Configuration

Outbound SSL connections from DataPower are configured in different ways, depending on the kind of connection being made: a simple backend SSL connection, or a dynamically specified connection to a URL either as a dynamic backend or for SSL connections initiated from XSLT.

Simple Backend Connections

For simple connections to the backend, a Crypto Profile is specified that contains all the SSL configuration information required. This Crypto Profile is specified slightly differently depending on the type of service. For instance, the XML Firewall uses a Crypto Profile object for connections to the backend known as the “SSL Client Crypto Profile,” as shown in Figure 18-18.



The screenshot shows a configuration window titled "Back End". It contains several fields and buttons:

- Server Address:** A text box containing "192.168.10.149" and an asterisk icon.
- Server Port:** A text box containing "9084" and an asterisk icon.
- SSL Client Crypto Profile:** A dropdown menu showing "MyCryptoProfile", followed by a "+" button and an ellipsis button.
- Response Type:** A dropdown menu showing "SOAP".
- Response Attachments:** A dropdown menu showing "Strip".

Figure 18-18 SSL Client Crypto Profile.

Recall that because the XML Firewall initiates the connection, it is the SSL client, and that is why the object is called an “SSL Client Crypto Profile.” At first, it might seem backward to call the connection to the backend the “Client Profile,” but it’s correct!

Because an XML Firewall configuration can use only HTTP or HTTPS and only the server hostname or IP address is specified, this Crypto Profile is used when connecting to the backend server, and the definition of this Crypto Profile results in the call being made using HTTPS instead of plain HTTP.

In the Multi-Protocol Gateway, outbound calls are also controlled by specifying a Crypto Profile; this time under the “Back side settings” for the proxy. This configuration is slightly different for each of the services. The configuration for the Multi-Protocol Gateway is shown in Figure 18-19.

Back side settings

Backend URL
 *

MQHelper TibcoEMSHelper WebSphereJMSHelper
 IMSCoconnectHelper

User Agent settings

Match	Property
<p>Note: To edit the User Agent, please access via the XML Manager above.</p>	

SSL Client Crypto Profile
 + - ...

Figure 18-19 SSL Client Crypto Profile for the Multi-Protocol Gateway.

Here, although the connection can use a number of protocols, the SSL Client Crypto Profile is only used if the backend URL is explicitly specified as https. (SSL for MQ is defined separately, as shown later in this chapter.)

For the Web Service Proxy, the definition is similarly under “Back side settings,” except that here it uses an SSL Proxy profile, as shown in Figure 18-20.

Back side settings

Selecting static-from-wsdl creates a static backend url by rewriting the wsdl:service address using the remote endpoint rewrite at configuration and refresh time.

User Agent settings

SSL Proxy
 + - ...

Figure 18-20 SSL Proxy Profile for the back side of a Web Service Proxy.

The SSL Proxy Profile specified is used only for outbound connections to the backend, so it is enough for it to only specify a forward or client Crypto Profile, unless that connection will require a client certificate for mutual authentication.

Dynamically Allocated Outbound SSL Connections

In all configurations, it is also possible to specify an SSL configuration using the User Agent in the XML Manager. The User Agent contains a special policy, called an SSL Proxy Profile Policy, where specific SSL Proxy Profiles can be configured to match specific URLs for outgoing connections. This is a powerful facility—different outbound calls from the same service could potentially use different SSL Proxy Profiles depending on where they are calling to. For instance, consider the configuration shown in Figure 18-21.

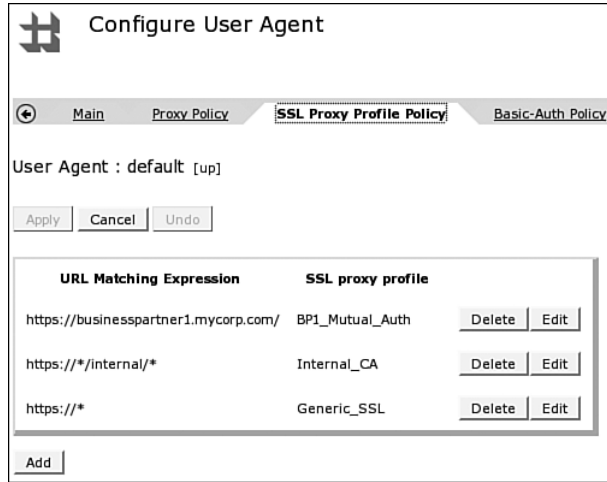


Figure 18-21 SSL Proxy Profile Policy within the User Agent.

Figure 18-21 shows three different SSL Proxy Profiles, which will be used depending on the outgoing URL. If the connection is made to `https://businesspartner1.mycorp.com`, the proxy profile called `BP1_Mutual_Auth` will be used. This profile is likely configured to provide a client certificate to authenticate to the business partner. If the connection is made to any URL where the URL stem (or URI) begins with `/internal/*`, it will use the SSL Proxy Profile called `Internal_CA`. It is likely that this profile contains a validation credential that trusts an internal certificate authority of some sort. Finally, all remaining connections use the SSL Proxy Profile called `Generic_SSL`, which likely contains some well-known trusted signers.

Certificate Revocation Lists

One of the problems with certificates is that, once a certificate is issued, it is very hard to take it away again. Since the certificate is signed by the signer, and validation of the certificate is based on the cryptographic principles described in this chapter, the validation will always succeed if the signature is valid. If a CA has mistakenly issued a certificate, there is no way for it to “un-sign” it. This is why CAs publish Certificate Revocation Lists (CRLs).

A CRL is simply a list of specific certificates that should no longer be considered valid, published in a format defined in RFC3280. DataPower can be configured to retrieve and use these CRLs. To create a policy for retrieving CRLs, click **Objects**→**Crypto**→**CRL Retrieval**. DataPower supports retrieving the CRLs using HTTP and LDAP, optionally encrypting the connection to the CRL server using SSL. Figure 18-22 shows an example of retrieving a CRL every four hours over HTTP from a server called `internal-crl-server`.

The validation credentials shown in Figure 18-22 are used to validate the signature of the CRL itself, thus ensuring that only a trusted CRL will be used.

Adding new CRL Update Policy property
of **CRL Retrieval**

Policy Name	local_ca_retrieval *
Protocol	HTTP *
CRL Issuer Validation Credentials	Internal_CA (Crypto Validation Credentials) + ... *
Refresh Interval	240 minutes *
Cryptographic Profile	
Fetch URL	http://internal-crl-server:49102 *

Save Cancel

Figure 18-22 CRL retrieval using HTTP.

Device Certificate

Each DataPower appliance has an SSL certificate used for encrypting access to the device over the Web management and XML management ports (ssh uses a different type of key). By default, every appliance uses the same certificate, which is shipped with the appliance. This certificate is signed by a DataPower SSL CA which, in turn, is signed by a DataPower Root CA. However, this certificate should not be used for production systems. The same certificate, and corresponding private key, ship with *every single DataPower appliance*. This means that the private key for the certificate is not very private at all!

Every DataPower customer should use their own SSL certificate for the appliance. This can be done in the default domain, by selecting System Control from the Control Panel in the WebGUI, and scrolling down to the section entitled “Generate Device Certificate,” shown in Figure 18-23.

Generate Device Certificate

Common Name (CN) 192.168.10.24 *

Generate Self-Signed Certificate ☒ on ☐ off *

Generate Device Certificate

Figure 18-23 Generating the Device Certificate.

Figure 18-23 shows the use of a self-signed SSL certificate; the device generates one with the requested common name, and creates an SSL proxy profile containing the certificate. This SSL proxy profile can then be set on the Web management, under Network→Management→Web Management Service, on the advanced tab, under Custom SSL Proxy Profile (likewise for XML management).

If you choose not to generate a self-signed SSL certificate, instead a private key and a Certificate Signing Request (CSR) will be generated to send to your CA for signing. These are saved to the temporary directory.

TIP — CAREFUL WITH DEVICE CERTIFICATES!

The two most important things about the device certificate are that you should not use the default certificate that ships on the device itself, and that you should be careful when changing it! If you mistakenly select an SSL proxy profile that is incorrectly configured, it is possible to lock yourself out of the Web management GUI entirely. In this situation, you would have to use the CLI to rectify the problem.

Advanced SSL Usage

The previous parts of this chapter gave enough theory and practical instructions to configure SSL for the majority of DataPower SSL use cases. This section goes into some of the more advanced functionality available.

Crypto Tools

The DataPower appliance ships with a set of crypto tools that allow you to generate keys and certificates right on the box itself. The crypto tools can also import and export crypto objects such as certificates (although it cannot export a copy of the private keys, unless your appliance is equipped with a Hardware Security Module (HSM), in which case it can export the private keys in a special HSM encrypted format which can later be imported). The crypto tools key generation page is shown in Figure 18-24.

This allows you to generate a public/private key pair along with a certificate signing request containing the public key. The certificate signing request can then be sent off to an internal or external (such as Verisign) certificate authority for signing. Alternatively, the crypto tools can generate a self-signed certificate, where the private key of the pair is used to sign the certificate, which can be useful.


The tools can also automatically create configuration objects for the key and certificates they generate, saving you the trouble of having to manually create these objects.

TIP — EXPORT THE PRIVATE KEYS!

When you generate your public-private key pair, make sure to export the private key, either to the temporary directory of the device (from which you should remove it and store it somewhere securely offline) or via the HSM if your appliance is equipped with one.

When you do export the keys, it is vital that there is a standard secure process defined for doing so, and that they are kept track of properly, with an audit trail. These keys are the keys to the kingdom—if someone gets hold of them, they can pretend to be your server! Above all else, avoid using methods such as an employee or a consultant putting the private key on a USB memory stick to move it from one computer to another—chances are that it will not be removed, and their child will take the memory stick to school because it also contains her homework, and then the school's geek squad will get hold of it...

If you do not export the private key at this time, you will be unable to retrieve it from the appliance in the future—the only solution will be to delete and re-create it, which means you would need to re-create the corresponding public key and get the new certificate signed, and so on. This could be very bad if you need to replace the appliance, or share the private keys in future for load balancing purposes!



Crypto Tools

Generate Key

Export Crypto Object

Import Crypto Object

Clone HSM Key Wrapping Key

Generate Key

LDAP (reverse) Order of RDNs

onoff

Country Name (C)

GB

State or Province (ST)

Locality (L)

Organization (O)

IBM

Organizational Unit (OU)

ISSW

Organizational Unit 2 (OU)

Software Group

Organizational Unit 3 (OU)

Organizational Unit 4 (OU)

Common Name (CN)

Self Signed Key 3 *

RSA Key Length

4096 bits

File Name

Validity Period

365 days

Password

Confirm Password:

Password Alias

Private Key Exportable via hsmkwk

onoff

Export Private Key

onoff

Generate Self-Signed Certificate

onoff

Export Self-Signed Certificate

onoff

Generate Key and Certificate Objects

onoff

Object Name

selfsigned

Generate Key on HSM

onoff

Using Existing Key Object

Generate Key

Figure 18-24 Crypto tools key generation.

SSL and FTP

The specification for using SSL with FTP is somewhat more complicated than with other protocols. This is in part because the FTP protocol itself is more complicated—it uses multiple TCP sockets, with a single long-lived “control connection” used for authentication and sending commands, and many “data connections” to actually send data such as directory listings and actual files. RFC4217 describes how SSL should be used with FTP, and DataPower implements this RFC.

Note that this discussion is explicitly about FTPS—FTP over SSL—and not “SFTP” with which it is commonly confused. SFTP is an “FTP-like” protocol that uses Secure Shell (SSH) to transfer files.

One of the big differences between FTPS and HTTPS is that SSL with FTPS is explicit. That is, with HTTP you simply specify the URL to be “https” and the client will automatically (implicitly) know to negotiate SSL, whereas with FTP, you establish a TCP socket without encryption and then you send a command to ask for negotiation of SSL. Furthermore, because SSL works at the socket level, data connections will need to be negotiated separately.

Figure 18-25 shows the configuration page for an FTP Server FSH.

Configure FTP Server Front Side Handler	
Main Virtual Directories	
FTP Server Front Side Handler	
Apply Cancel	
Name	FTPS_FSH *
Admin State	<input checked="" type="radio"/> enabled <input type="radio"/> disabled
Comments	
Local IP Address	0.0.0.0 Select Alias *
Port Number	21 *
Filesystem Type	Virtual Ephemeral
Default Directory	/
Maximum Filename Length	256
Access Control List	(none) + ...
Require TLS	<input checked="" type="radio"/> on <input type="radio"/> off
SSL Proxy	Generic_SSL + ...
Password AAA Policy	(none) + ...
Certificate AAA Policy	(none) + ...
Allow CCC Command	<input type="radio"/> on <input checked="" type="radio"/> off
Passive (PASV) Command	Require Passive Mode
File Transfer Data Encryption	Require Data Encryption
Allow Compression	<input checked="" type="radio"/> on <input type="radio"/> off
Allow Unique File Name (STOU)	<input type="radio"/> on <input checked="" type="radio"/> off
Idle Timeout	0 seconds
Response Type	No Response
Temporary Storage Size	32

Figure 18-25 FTP Server FSH.

On this configuration page, there are a number of SSL-related settings and we will cover them from top to bottom. The first of these is “Require TLS.”⁴ If this option is set to on, no FTP

⁴ TLS stands for Transport Layer Security and is the new name of the SSL protocol. TLS 1.0 was based on SSL 3.0, and the current version of TLS is 1.1. However, many people still refer to TLS as SSL, and the two can be reasonably interchanged in most contexts. RFC4346 provides a list of the differences between TLS1.0/SSL3.0 and TLS1.1, which consist of some minor security improvements and small clarifications of the specification.

commands will be accepted over a non-SSL connection except the FTP AUTH TLS command used to initiate an SSL negotiation. This will ensure that SSL is used for all FTP command data. The SSL Proxy setting chooses which SSL Proxy Profile will be used with this FSH. The setting to Allow CCC Command determines whether clients, after they have negotiated an SSL connection, will be allowed to fall back to clear text for the control connection (CCC stands for Clear Control Connection). Finally, the File Transfer Data Encryption option can be set to allow, disallow, or require encryption of the data connections as well as the control connection. Using these configuration options, it is possible to ensure that 100 percent of the FTP traffic to and from the appliance goes over SSL.

In a similar manner, Figure 18-26 shows the FTP Client Policies tab of the user agent.

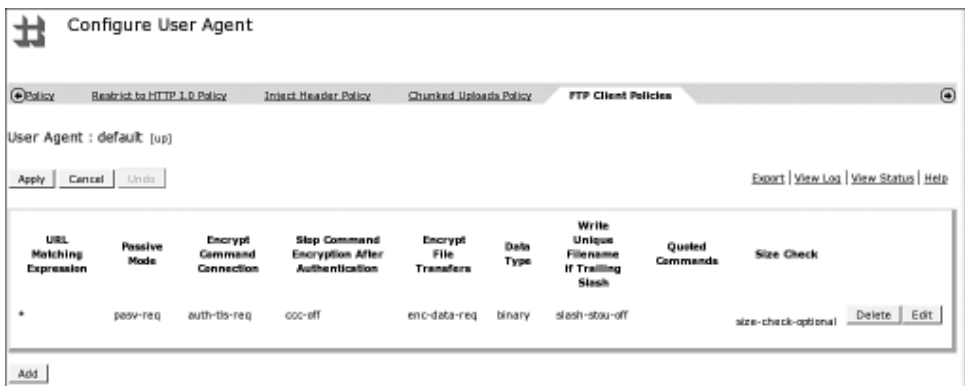


Figure 18-26 FTP client policies.

The FTP client policies control outbound FTP connections, and enable the configuring of exactly the same SSL functionality as on the server side, although the names used are slightly different.

When using SSL with FTP, you should always consider two things. First, because the FTP protocol uses multiple sockets for data, whether file listings or files themselves, each data connection requires a separate SSL handshake. Because of this, it is absolutely vital to use both client and server side SSL session identifier caching. Otherwise each and every socket will entail a full PKI handshake, which will not be good for performance as things scale up!

Secondly, using SSL for FTP will of necessity break the support that some firewalls have for FTP. These stateful packet filtering firewalls have explicit support for FTP, whereby they will monitor the control connection and dynamically open ports in the firewall for data connections as needed. This will not work if the data connection is encrypted, because the firewall will not have the key to decrypt the traffic! For this reason, it is sometimes useful to allow the CCC command so that a client can negotiate the initial SSL session to perform cryptographic authentication, and can still encrypt all data connections, but can use the session long control connection in plain text so that the firewall will be able to dynamically open ports.

SSL and MQ

SSL is used with MQ as a standard method for performing encryption of traffic and cryptographic authentication at a server level. However, configuration of SSL for MQ on DataPower is slightly more complex and requires further explanation.

Figure 18-27 shows part of the definition of an MQ queue manager object.

SSL Key Repository	cert:///mqkeys.pem	Upla
SSL Cipher Specification	TLS_RSA_WITH_AES_128_CBC_SHA	
Convert Input	<input checked="" type="radio"/> on <input type="radio"/> off	
Automatic Retry	<input checked="" type="radio"/> on <input type="radio"/> off	
Retry Interval	1	seconds
Reporting Interval	1	seconds
Alternate User	<input checked="" type="radio"/> on <input type="radio"/> off	
Local Address		
XML Manager	default	+ ... *
SSL proxy profile	Generic_SSL	+ ...

Figure 18-27 An MQ Queue Manager object.

This object has two separate sets of SSL configuration: the SSL Proxy Profile field we know already, and a the slightly older concept of an SSL Key Repository and SSL Cipher Specification. This is because the behavior of the MQ connection depends on how the queue manager object is used.

If the queue manager definition is called by a service that uses an MQ URL to designate its remote destination, the specified SSL Proxy Profile will be used and everything will work exactly as expected. This is the case, for instance, with a Multi-Protocol Gateway. If on the other hand the queue manager object is used by any other type of service (such as the older MQ Gateway or MQ Proxy objects), they will instead use the defined key repository on the queue manager object definition; although these services are rarely used nowadays, and in most cases the configuration should be as simple as shown.

The SSL key repository is a pointer to a key database file and corresponding stash file that must be uploaded to the device. Moreover the stash file must be named exactly the same as the actual key database except that the file extension must be .sth. If the name of the stash file is not the same in this way, the key database will not be able to be opened.

Whether the SSL configuration is specified as an SSL Proxy Profile or as an MQ-specific key repository, it is absolutely critical that the cipher specification and SSL options are compatible with the SSL configuration on the MQ side. For reference, Table 18-1 contains a mapping between the MQ SSL ciphers and DataPower Crypto Profile options.

Table 18-1 Mapping of MQ SSL Ciphers to DataPower Crypto Profile Options

MQ SSL Cipher	SSL Cipher Specification	SSL Options
NULL_MD5	NULL-MD5	OpenSSL-default+Disable-TLSv1
NULL_SHA	NULL-SHA	OpenSSL-default+Disable-TLSv1
RC4_MD5_EXPORT	EXP-RC4-MD5	OpenSSL-default+Disable-TLSv1
RC4_MD5_US	RC4-MD5	OpenSSL-default+Disable-TLSv1
RC4_SHA_US	RC4-SHA	OpenSSL-default+Disable-TLSv1
RC2_MD5_EXPORT	EXP-RC2-CBC-MD5	OpenSSL-default+Disable-TLSv1
DES_SHA_EXPORT	DES-CBC-SHA	OpenSSL-default+Disable-TLSv1
RC4_56_SHA_EXPORT1024	EXP1024-RC4-SHA	OpenSSL-default+Disable-TLSv1
DES_SHA_EXPORT1024	EXP1024-DES-CBC-SHA	OpenSSL-default+Disable-TLSv1
TRIPLE_DES_SHA_US	DES-CBC3-SHA	OpenSSL-default+Disable-TLSv1
TLS_RSA_WITH_AES_128_CBC_SHA	AES128-SHA	OpenSSL-default
TLS_RSA_WITH_AES_256_CBC_SHA	AES256-SHA	OpenSSL-default
AES_SHA_US	AES128-SHA	OpenSSL-default

When Signing Isn't Enough

Recall that with SSL, trust is established by checking whether the digital signature of a certificate is signed by a trusted third party that we have chosen to trust. Under DataPower we configure who we trust by creating a validation credential object.

What happens if this is not fine-grained enough? For instance, what if we want to confirm that not only does the peer have a certificate that is signed by someone we trust, but also that the signed information contains some values that are relevant and important to us? Well, because security is part and parcel of the DataPower appliance, we can examine the signed data and make our decision based on this.

As described in Chapter 16, “AAA,” it is possible as part of a AAA policy to choose as an identification method the “Subject DN of the SSL Certificate from the Connection Peer.” What does this mean? Well, it means that, after the SSL handshake has been negotiated and session established, based on cryptographic trust, we can then look at the data in the signed certificate and make our decision based on whether or not the distinguished name or DN of the certificate is one we want to allow. Alternatively, we could use the techniques shown in Chapter 17, “Advanced AAA,” to make our identity extraction use a custom stylesheet and examine any of the signed data that we so choose.

The SSL Proxy Service

What's that? Haven't we already talked about SSL Proxies? Well, no—earlier we discussed SSL Proxy *Profiles*, but the SSL Proxy *Service* is a completely separate kind of service that deserves mention. Figure 18-28 shows an SSL Proxy Service configuration.

Figure 18-28 The SSL Proxy Service.

The SSL Proxy Service is used to provide a form of tunnel through which connections can be made to a backend server. At its simplest level, this service can provide SSL forwarding; that is, if you connect via SSL to this service, it will connect via SSL to its remote host and any data that you send over your connection to the proxy service will be sent on to the remote host. This in and of itself is likely not all that useful. A rare but valid use case for this might be if the connecting server was unable to support client certificates, so we could add client certificate authentication on the outbound connection to the backend of the proxy.

A more powerful use of the SSL Proxy Service is the fact that the connections do not *have* to utilize SSL. Of course if you use no SSL at all, you should simply use the TCP proxy service, but if you want *either* the incoming *or* the outgoing connection to use SSL, the SSL Proxy Service is perfect. This functionality is similar to the acclaimed open source `stunnel` tool.

The two use-cases are therefore

1. To proxy an inbound SSL connection to an outbound TCP connection, allowing transparent addition of SSL capabilities to a non-SSL service
2. To proxy an inbound TCP connection to an outbound SSL connection, allowing encryption of a call to a backend service where the client is not capable of SSL

Use case 1 is a simple way to quickly upgrade an existing HTTP service to work as HTTPs. Of course, hard-coded links to `http://` URLs will not work, meaning that this service is not really suitable for proxying complex Web applications, but if the requirement is simply to add an SSL frontend, then this is possible. Indeed, this could be used for more than just HTTP—any plain TCP service could be fronted with SSL, so for instance you could enable your LDAP server to work as LDAPS even if it does not support this natively.

To configure use case 1, you would configure the SSL Proxy Profile with an SSL Direction of Reverse and add the certificate you want the service to use as an identification credential in the associated Crypto Profile. Of course, you could also configure the profile with validation credentials and support or enforce mutually authenticated SSL should you desire.

Use case 2, on the other hand, provides a powerful mechanism to encrypt connections to servers where the client is not capable of SSL. For example, imagine you have an LDAP server that accepts only connections using LDAPS, but one of your legacy applications is unable to use SSL for its LDAP client connections. Rather than modifying the legacy application, you could use an SSL proxy service on the DataPower appliance, and point the LDAP client at the port on the appliance, which would then proxy the connection to the backend using SSL.

To implement use case 2, the SSL Proxy Profile would need to be set with an SSL Direction of Forward. A validation credential could be configured to validate the certificate of the backend server. Again, this connection could be made to use mutual authentication.

The Mutually Authenticated Self-Signed SSL Tunnel

Finally, we examine and describe a very important usage pattern that is an extension of that described earlier in the “The SSL Handshake” section.

The scenario is this: We have two DataPower appliances that want to communicate with each other securely. These appliances use a mutually authenticated SSL connection to ensure that all traffic is encrypted and that each appliance can be certain of the identity of the other appliance. To severely limit the number of other nodes that are considered “trusted,” we use two separate self-signed certificates.

The advantage of using a self-signed certificate in this manner is that, with a self-signed certificate, there is exactly one, and only one, certificate that has been signed by the signer—and that is the certificate itself. If our validation credential on either side contains no certificates apart from the self-signed certificate of the peer, no one can complete an SSL handshake with us unless they hold the private key to that exact certificate. Because DataPower sensibly does not include any certificates into a validation credential by default, there should not be as much room for error as there is when configuring this pattern on other systems!

Figure 18-29 shows the configuration used here.

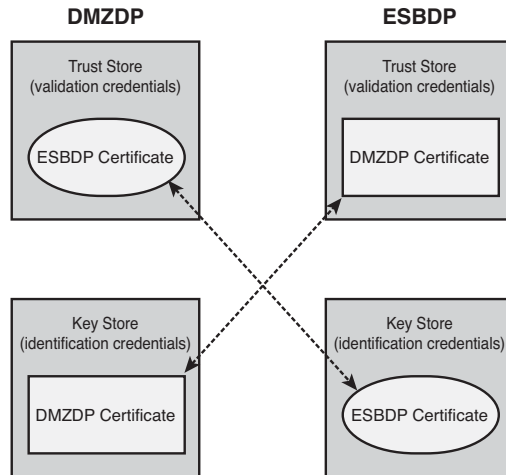


Figure 18-29 The mutually authenticated self-signed SSL tunnel.

As you can see from Figure 18-29, the validation credentials for DMZDP contain exactly one certificate, which is the self-signed certificate of ESB DP, and vice versa—the validation credentials for ESB DP contain exactly one certificate, which is the self-signed certificate of DMZDP. Note that, although the certificate itself is shared, the private key relating to the certificate remains private—only DMZDP has DMZDP's private key, and only ESB DP has ESB DP's private key. Because of this, even if someone else were to somehow get hold of the certificate, he would still be unable to connect because he would not have the corresponding private key.

This powerful technique allows us to link two arbitrary nodes in such a way that we can have absolute cryptographic trust that there is exactly one, and only one, node which will be able to communicate—and that is the configured SSL peer. Indeed this same pattern could be extended by sharing the certificates with a number of nodes, all of whom would be allowed to connect; possession of the private key to the client SSL certificate itself becomes the token by which authorized nodes prove that they should be allowed to connect.

TIP—POINT-TO-POINT SECURITY

People often get excited by the idea of using WS-Security. There is a lot of value in using parts of the specification, and DataPower contains one of the most advanced and current implementations of the specifications available at time of writing. However, if your goal is to simply secure a point-to-point connection between client and server, with no untrusted intermediaries, and provide privacy and integrity for the messages sent over that connection—a mutually authenticated SSL tunnel with a very limited trust domain is a superb way to implement this!

Of course, if your requirements are such that you are routing messages through multiple untrusted intermediaries and have actual requirements for message level encryption and signatures, you may have a valid WS-Security use case, in which case you should read Chapter 19.

Troubleshooting SSL

As this chapter so far has shown, SSL is a powerful mechanism for encryption and cryptographic authentication. However, what happens when things go wrong? How can we diagnose, debug, and repair SSL problems on DataPower?

What Can Go Wrong?

SSL is powerful, but like most powerful concepts, it is complex and thus configuration becomes error prone. This section presents a number of common pitfalls.

Client Authentication Is Optional

The option to say that client authentication is optional is a specific form of double negative. If you say yes, meaning that you want client authentication to be optional, this will mean that clients will *not* have to authenticate (provide a client SSL certificate) to connect. This might seem like a simple problem, but the tricky thing here is that if this is set incorrectly, the connection will *still work*! This is a good example of the general problem that it is not enough to configure something such that it works—you have to configure it explicitly to be secure. In this instance, if the client is configured to use an SSL certificate, it will be requested as part of the SSL handshake and will be sent and used, but if the client does not have an SSL certificate, it still can connect and the user will see no difference.

Incorrect Cipher Suites

Recall that as part of the SSL handshake, the peer that initiates the connection (and thereby becomes known as the client) sends a list of ciphers that it supports, and the server side then chooses which of those ciphers to use. Although the majority of SSL clients and servers are configured correctly, it is still possible that you will connect to a server or client that will choose a completely inappropriate cipher to use. It is, therefore, important to decide what cipher suites are acceptable to your business, and only specify those on the configuration. If you do not, it is possible that while your system will work, it will not be as secure as you believe.

This specific issue was hit unknowingly by many consumers a few years back, when a well-known Internet browser contained a bug causing it to use only weak export quality encryption. Web sites that had been correctly configured to use only strong encryption “broke” in that the browser was unable to communicate with them because they refused to accept weak encryption. However, a large number of public sites were still accessible over HTTPS, because they had never been configured to use only the high-quality encryption and, therefore, graciously stepped down to the low-quality weak encryption!

Certificates for Proxy

Recall that in a proxy scenario, which is the most common form of DataPower deployment, there are two SSL connections: one from the client to the proxy, and a second one from the proxy to the backend server. It is easy to misunderstand which certificate should be placed where. In a proxy

scenario, the certificate that would usually have been used by the Web server, or whatever the backend server happens to be, needs to be moved over to the DataPower appliance, along with its private key so that it can be used for encryption. This means that for the original backend server, a new certificate should usually be generated; it is not a good practice to use the same certificate at two layers. Moreover the hostname that was originally used for the certificate should be changed to point to the DataPower appliance, so that it can truly proxy (pretend to be the original server).

Debugging SSL

When things do go wrong, how can you go about working out exactly what is wrong in order to fix it?

Debug Logs

Because the SSL handshake happens before any actual application level traffic is sent over the socket, the Probe cannot be used to debug problems with SSL. However, if you turn on a high level of debug log messages, this will show any errors encountered in the SSL configuration. These debug messages can be extremely helpful in not only showing that the SSL handshake has failed, for instance, but also explaining why exactly it failed and what should be done to rectify the problem.

Packet Capture

If there is a problem with the SSL handshake, one of the most useful ways to debug this is to take a packet capture of the beginning of the handshake. When fed through a high-quality packet capture interpretation tool, such as the excellent freeware Wireshark, the packet capture shows the SSL handshake as it happens, right up until the moment of first encryption. (It is possible to decrypt more if you have the private key, of course, although that is out of scope of this book.)

Client Side Debugging

Many SSL clients have various facilities for SSL debugging. Of note, the cURL utility is capable of producing an extremely detailed debug log showing the various stages of the handshake, the certificates used, and so on. In order to enable SSL debugging in cURL, you can use the `-v` parameter to request verbose messages; this will display the full SSL handshake.

An example of the level of debugging is shown in Listing 18-1. This is the output of the cURL tool being run with `-v` to connect over SSL to the IBM home page, www.ibm.com.

Listing 18-1 Curl Provides Detailed SSL Debugging Information

```
[user@laptop SSL]$ curl -v https://www.ibm.com
* About to connect() to www.ibm.com port 443
*   Trying 129.42.60.216... connected
* Connected to www.ibm.com (129.42.60.216) port 443
* successfully set certificate verify locations:
*   CAfile: /etc/pki/tls/certs/ca-bundle.crt
*   CAPath: none
* SSLv2, Client hello (1):
SSLv3, TLS handshake, Server hello (2):
SSLv3, TLS handshake, CERT (11):
SSLv3, TLS handshake, Server finished (14):
SSLv3, TLS handshake, Client key exchange (16):
SSLv3, TLS change cipher, Client hello (1):
SSLv3, TLS handshake, Finished (20):
SSLv3, TLS change cipher, Client hello (1):
SSLv3, TLS handshake, Finished (20):
SSL connection using RC4-MD5
* Server certificate:
*   subject: /C=US/ST=North Carolina/L=Research Triangle
Park/O=IBM/OU=Events and ibm.com infrastructure/CN=www.ibm.com
*   start date: 2008-02-20 00:28:07 GMT
*   expire date: 2009-05-21 23:28:07 GMT
*   common name: www.ibm.com (matched)
*   issuer: /C=US/O=Equifax/OU=Equifax Secure Certificate Authority
* SSL certificate verify ok.
> GET / HTTP/1.1
> User-Agent: curl/7.15.5 (i686-redhat-linux-gnu) libcurl/7.15.5
OpenSSL/0.9.8b zlib/1.2.3 libidn/0.6.5
> Host: www.ibm.com
> Accept: */*
>
< HTTP/1.1 302 Found
< Date: Sun, 15 Jun 2008 03:31:27 GMT
< Server: IBM_HTTP_Server
< Location: http://www.ibm.com/
< Content-Length: 203
< Content-Type: text/html
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www.ibm.com/">here</a>.</p>
</body></html>
* Connection #0 to host www.ibm.com left intact
* Closing connection #0
* SSLv3, TLS alert, Client hello (1):
[user@laptop SSL]$
```

Note how each step of the handshake is shown, and then the details of the server's certificate and its signer are displayed. Note also this graphic example of the SSL handshake and connection being completed *before* any application traffic (the HTTP request and response) are ever sent over the wire.

Summary

In this chapter, we examined SSL in depth, exploring its configuration on the DataPower appliance and how extremely valuable and critical it is to the basic function of not just DataPower, but all Internet traffic.

The general cryptographic concepts presented will serve as a base for the equally valuable information in the next chapter, which discusses Web Services security.