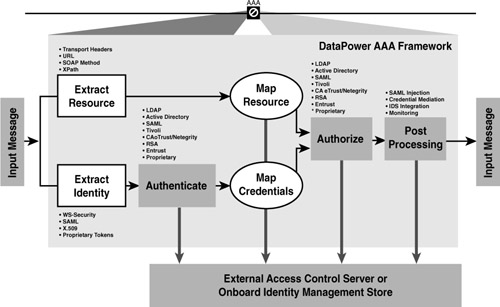
**Chapter 17: Advanced AAA**

Add a note hereAs [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926), [“AAA,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) shows, the DataPower AAA framework is flexible and powerful, allowing you unparalleled control over which users or systems can access your services. This chapter goes through some examples of this at work, including customization of the framework and integration with IBM Tivoli Access Manager (TAM).

Add a note hereCustomization is an inherent part of the DataPower device and isn’t something to be afraid of. While there are many out-of-the-box options that cover the vast majority of customer requirements, especially in a complex area such as security, there is no way that everything can be covered. The AAA framework enables you to customize to a granular level to integrate with even the most complex of solutions and fulfill even the most difficult requirements.

**Add a note here****Customizing the AAA Runtime Process**

Add a note hereThe AAA runtime described in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) can be customized at almost every step. (The exception is resource extraction, but this can be fixed by a subsequent customized resource mapping step.) The steps of the AAA flow are summarized in Figure 17-1. To write a AAA customization stylesheet, you need to know what the inputs and outputs of each stage need to look like.

[](javascript:PopImage('IMG_402','http://images.books24x7.com/bookimages/id_30903/17fig01_alt.jpg','980','602'))  
Add a note hereFigure 17-1: AAA flow summary.

**Add a note here****XML in the AAA Flow**

Add a note hereThe AAA flow works by passing an XML tree from one step to the next; the XML tree contains information about what happened in the earlier stages. It is built automatically by the out-of-the-box AAA methods; some of these XML trees were shown in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) as the contents of the debug variables for the various stages of AAA processing.

Add a note hereWhen customizing the AAA flow, our own stylesheets need to work with this XML tree and create the relevant output for the next stage. This may seem somewhat complex, but in truth is no different to writing any other XSL; our processing must simply transform the input tree from one stage to the output tree of another, and by doing so, it communicates decisions to DataPower regarding authentication, authorization, and so on.

Add a note hereThis processing is best demonstrated by example, so we will walk you through each stage and show an example of processing at each level. It is highly likely that when you do need to customize the AAA process, you will have to customize only one or two stages; our approach, therefore, enables you to pick the pieces you need to use to customize in your specific environment.

**EI—Extract Identity**

Add a note hereThe EI stage has many out-of-the-box options that cover the vast majority of user requirements. Indeed, it is difficult to think of a realistic requirement that is not already covered! Of course there are real-world requirements, but they do not make for an easy demonstration of the functionality, so this example may appear somewhat contrived; we can only apologize for the huge array of supported out-of-the-box use cases!

Add a note hereThe example we use is that of an XML document that contains a username and a password in a specific node. For most similar use cases, we would use the out of the box processing called Token Extracted from the Message, which allows you to specify an XPath expression to extract the required string. However, in this instance the username field needs to be modified before passing to the authenticate step, and therefore, we need to use custom processing.

Add a note hereThe input for the EI stage custom processing is the entire XML message passed from the input context to the AAA action. The entire request is available, including protocol headers, through DataPower extension functions. It may be that other processing has taken place before the AAA action; for instance, the client XML message may be encrypted and a Decrypt action was needed, or perhaps form-based login was used in a Web application and a transformation was required to convert it from HTTP to XML. For our example, however, no previous processing has taken place, and the request document appears as shown in Listing 17-1.

Add a note hereListing 17-1: Request Document

Add a note here<?xml version="1.0"?>

<getAmount>

<authentication>

<username>alice</username>

<password>mysecret</password>

</authentication>

<data>

<values/>

</data>

</getAmount>

Add a note hereClearly the username and password need to be extracted from the request; however, in this instance, this is not enough, because our authentication server requires usernames to be in the form of an email address to uniquely identify specific users. (There is more than one alice in the world, for example.) The domain part of the username is passed in using an HTTP header, AuthDomain, the value of which needs to be appended to the supplied username in order to form an email address.

Add a note hereThis is accomplished using the stylesheet shown in Listing 17-2. If you are not familiar with XSLT, a wealth of information about developing custom stylesheets can be found in [Part VI](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4077#4077) of this book, “[DataPower Development](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=4077#4077).”

Add a note hereListing 17-2: Identity Extraction Stylesheet—IE.xsl

Add a note here<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:dp="http://www.datapower.com/extensions"

exclude-result-prefixes="dp">

<xsl:output method="xml"/>

<xsl:template match="/">

<!-- Use /\*/ to match various requests -->

<xsl:variable name="username"

select="/\*/authentication/username/text()"/>

<xsl:variable name="password"

select="/\*/authentication/password/text()"/>

<xsl:variable name="domain"

select="dp:request-header('AuthDomain')"/>

<username><xsl:value-of

select="concat($username,'@',$domain)"/></username>

<password sanitize="true"><xsl:value-of

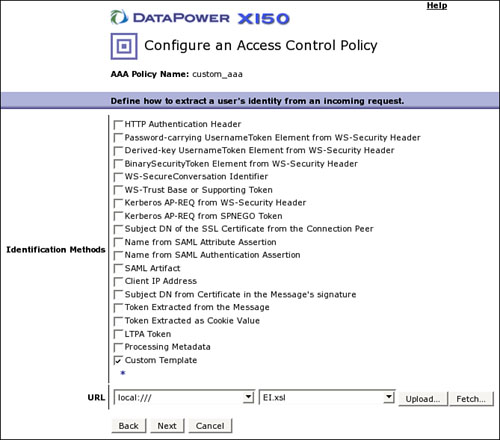
select="$password"/></password>

</xsl:template>

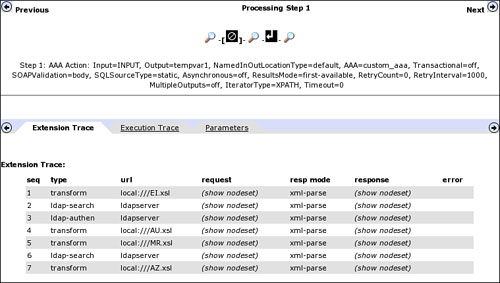
</xsl:stylesheet>

Add a note hereThis stylesheet processes the input document directly, searching for the username and password fields. It also retrieves the value of the AuthDomain HTTP header. Finally, it outputs an XML structure containing username and password elements with the relevant values for authentication. This choice of output is not accidental; the expected output in this instance is two XML nodes—username and password—with no wrapper element.

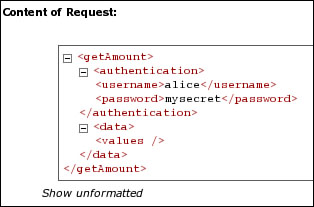
Add a note hereThe stylesheet is configured in the EI step of a AAA policy, as shown in Figure 17-2.

[](javascript:PopImage('IMG_403','http://images.books24x7.com/bookimages/id_30903/17fig02_alt.jpg','669','589'))  
Add a note hereFigure 17-2: Select a custom template and provide the XSL file.

Add a note hereWe can use the Probe as described in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) to examine the execution trace of the AAA action and see the XML nodeset on the input and output of each custom transformation. If we enable a Probe, submit a transaction, refresh the Probe, and click on that transaction, we are shown a step-by-step representation of the various stages of processing that the request went through. We can click on each stage, in turn, to get details; if we click on the AAA action in the policy, the first tab titled Extension Trace gives details of the custom processing in the AAA policy. As shown in Figure 17-3, each stage has a request and a response nodeset. We can click on the link next to each one to get details of the input or output nodeset for the next part of the processing.

[](javascript:PopImage('IMG_404','http://images.books24x7.com/bookimages/id_30903/17fig03_alt.jpg','759','430'))  
Add a note hereFigure 17-3: The processing as seen in the Probe.

Add a note hereFor instance, Figure 17-4 shows the input into the EI step in the Probe, retrieved by clicking the (show nodeset) link in the request column, which confirms that the input to the EI step is the full submitted XML message.

  
Add a note hereFigure 17-4: Input into the EI step.

Add a note hereThis is then transformed by our stylesheet, which also includes the value of the HTTP header AuthDomain, and the output is shown in Figure 17-5.

  
Add a note hereFigure 17-5: Output from the EI step.

Add a note hereThe output from our custom processing presents simply the extracted identity, which DataPower wraps in an XML structure and passes to the next step.

**Tip: Sanitize Passwords**

Add a note hereThe previous example shows a construct containing the password, and an attribute has been added to the password node, sanitize=“true”. By adding this special attribute, we are telling the appliance that the text contents of this node should be masked in any log messages that the device happens to display. For instance, even with debug level AAA logging enabled, the device replaces the actual password supplied with asterisks. This ensures that, when the logs are stored “off box” in a persistent manner (as they always should be in a production environment), the password information is not stored in an insecure location that is not directly under the access control of the device.

Add a note hereNote that this sanitization does not apply to the Probe, as should be clear from Figure 17-2. This should not be cause for concern because the Probe can only be enabled on the appliance itself, and even then only by a user with authority to modify the object being probed. In addition, the Probe should not be used in production (as the warning on the administrative panel when enabling the Probe states clearly).

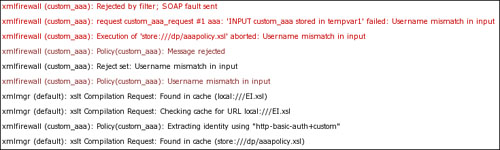
Add a note hereThe XML output from our EI stylesheet is used as the input to the AU step, so the exact content of the response is passed as the input. DataPower automatically wraps this in an XML structure identifying it as an entry inside an identity, and the entry has an attribute called type with the value custom and a second attribute called url showing where the custom stylesheet is stored. This information is shown in Figure 17-6, which shows the input into the AU step, again taken from the extension trace in the Probe.

[](javascript:PopImage('IMG_407','http://images.books24x7.com/bookimages/id_30903/17fig06.jpg','422','168'))  
Add a note hereFigure 17-6: Output from EI is also input to AU.

Add a note hereThe identity XML tree can have more than one entry node, depending on how many options were selected when defining the policy. Thus, if we were to select IP address as well as custom, we would have two entry nodes added by DataPower: one of type custom and the second of type client-ip-address, as shown in Figure 17-7.

[](javascript:PopImage('IMG_408','http://images.books24x7.com/bookimages/id_30903/17fig07.jpg','422','205'))  
Add a note hereFigure 17-7: It is possible to have more than one entry as the output from EI.

Add a note hereHowever, be aware that the contents can conflict with each other and lead to unexpected results. For instance, because we have created a username and password node, it is likely that our next step, AU, will be configured to use that username and password node to authenticate the user. If we were to *also* select HTTP basic authentication for EI, things would work just fine as long as *either* the custom stylesheet *or* the basic authentication header contained a valid username and password combination. However, if the user then provided *both* a HTTP basic authentication header *and* a username/password for our custom XSL to extract identity from, there would be two sets of username and password nodes. In that situation, because there is no way to tell which set to use, the device will use neither and simply deny the request, as shown in Figure 17-8 (taken from the DataPower log).

[](javascript:PopImage('IMG_409','http://images.books24x7.com/bookimages/id_30903/17fig08_alt.jpg','735','220'))  
Add a note hereFigure 17-8: Supplying identical identity nodes with two types will fail.

Add a note hereIncidentally, if the usernames are the same, at least processing continues to the next stage, although this may not be what you want, especially if a different password is specified for each!

**AU—Authenticate**

Add a note hereThe AU stage takes the input of the EI stage and processes the extracted identity such as to authenticate it; that is, to ensure in some manner that a valid set of credentials have been presented to prove that the peer entity supplying the credentials actually is the identity they claim to be.

**Warning: Security Programming Is Hard**

Add a note hereIt is easy to accidentally write an AU stylesheet that works and yet is completely insecure. Customization of the AAA process needs to be done with care and attention because this is security code that defines fundamental access control for your system, but, in particular, the AU stylesheet is easy to write badly and should be reviewed and validated many times.

Add a note hereAt the AU stage, your stylesheet must decide a basis on which to either trust or not trust the supplied credentials. If this trust is too weak, people will be able to abuse the trust to access resources they should not be able to access. For example, if the stylesheet simply looks at an HTTP header to determine whether the request comes from a specific trusted server, a malicious client could simply spoof this header and become authenticated. If on the other hand the header contains an encrypted signed short lifetime token containing a username, malicious clients would find it much harder to compromise.

Add a note hereAn important point for authentication in particular is that successful authentication is signified from an AU stylesheet by outputting any valid XML. Failed authentication is shown by outputting *no* XML at all. It is key to understand this; if your stylesheet outputs any valid XML, authentication is considered to have succeeded, even if the node set looks like this:

Add a note here<authentication>

<status>FAILED</status>

</authentication>

Add a note hereThis XML would actually be seen as successful authentication if output by a custom AU stylesheet! To signify authentication failure, you must output nothing.

Add a note hereSo, let’s take a look at an example AU stylesheet. This stylesheet takes as its input the username and password from the previous example. Recall that the username was made up in the form of an email address, and the domain of the email address was referred to as an authentication domain. Our goal in this AU stylesheet is to authenticate this user to an LDAP server, using a different base distinguished name depending on the domain provided.

Add a note hereThe LDAP server works by looking up the email address provided by the client and resolving that into a specific distinguished name; that distinguished name is then used to bind to validate that the password is correct. The AU.xsl stylesheet that performs this interaction with LDAP is shown in Listing 17-3.

Add a note hereListing 17-3: AU Stylesheet Communicating with LDAP—AU.xsl

Add a note here<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:dp="http://www.datapower.com/extensions"

exclude-result-prefixes="dp">

<xsl:output method="xml"/>

<xsl:template match="/">

<!-- Take the username password and domain from the incoming nodeset -->

<xsl:variable name="username"

select="/identity/entry[@type='custom']/username/text()"/>

<xsl:variable name="password"

select="/identity/entry[@type='custom']/password/text()"/>

<xsl:variable name="domain"

select="substring-after($username,'@')"/>

<xsl:choose>

<xsl:when test="$domain=('example.com' or 'somewhere.com')">

<xsl:variable name="SearchFilter"

select="concat('(&amp;(objectClass=ePerson)(emailAddress=',

$username,'))')"/>

<!-- make a SUB search for our email address to find the DN -->

<xsl:variable name="userid"

select="dp:ldap-search('ldapserver','389','','',

concat('o=',$domain),'dn',$SearchFilter,'SUB','','')"/>

<!-- if the search returned a distinguished name... -->

<xsl:if test="$userid/LDAP-search-results/result/DN/text()">

<!-- ...then try and bind as that distinguished name -->

<xsl:if test="dp:ldap-authen(

$userid/LDAP-search-results/result/DN/text(),

$password,'ldapserver:389')">

<!-- output XML if successful -->

<authenticated>yes</authenticated>

</xsl:if>

</xsl:if>

</xsl:when>

<xsl:otherwise>

<!-- OUTPUT NOTHING - AU fails -->

</xsl:otherwise>

</xsl:choose>

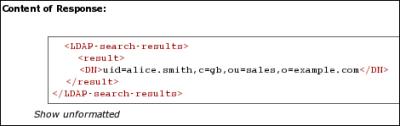
</xsl:template>

</xsl:stylesheet>

Add a note hereLet’s examine this processing in slightly more depth. This is an LDAP server with two base distinguished names: example.com and somewhere.com. To perform the initial search, we first build an RFC2254-compliant LDAP filter string that looks for an LDAP object with objectClass of ePerson (a common object class representing a user) and an email address equal to that supplied (which was built, recall, by adding the username to the AuthDomain HTTP header). This search filter is then used in an ldap-search extension function call, to contact the server at hostname ldapserver on port 389, and perform an LDAP search for a subset of the relevant base Distinguished Name and extract the DN field specific to that user. Figure 17-9 shows the input into the ldap-search call, again taken from the extension trace in the Probe.

[](javascript:PopImage('IMG_410','http://images.books24x7.com/bookimages/id_30903/17fig09_alt.jpg','595','177'))  
Add a note hereFigure 17-9: The input nodeset to the ldap-search call.

Add a note hereThis nodeset is built automatically by DataPower as a result of running the ldap-search extension function. As is clear from Figure 17-9, the actual connection to LDAP for the search is anonymous. If the server did not support anonymous binds for searching, we could specify a DN and password to use in the ldap-search function call. The search does not authenticate the user at all; rather it finds the attribute in the lookupAttribute node, in this case the DN, shown in Figure 17-10.

[](javascript:PopImage('IMG_411','http://images.books24x7.com/bookimages/id_30903/17fig10.jpg','470','149'))  
Add a note hereFigure 17-10: The nodeset containing the result of the LDAP search.

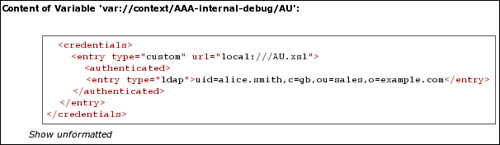
Add a note hereAfter we have the distinguished name, we then attempt to bind as that user with the password provided, by using the ldap-authen function call. The ldap-authen input parameters in this instance are shown in Figure 17-11.

[](javascript:PopImage('IMG_412','http://images.books24x7.com/bookimages/id_30903/17fig11_alt.jpg','506','180'))  
Add a note hereFigure 17-11: The nodeset built by ldap-authen.

Add a note hereAgain, this nodeset is automatically built by the ldap-authen function call. If this bind fails, the ldap-authen call returns no XML nodeset. If the bind is successful, however, the response of the ldap-authen call will contain the DN that was authenticated, depicted in Figure 17-12.

[](javascript:PopImage('IMG_413','http://images.books24x7.com/bookimages/id_30903/17fig12_alt.jpg','560','95'))  
Add a note hereFigure 17-12: The response nodeset from the ldap-authen call.

Add a note hereOur stylesheet simply needs to output any valid XML in order for authentication to have been deemed “successful.” However, it is often useful to be able to pass on a credential of some sort to the rest of the AAA policy, such that it can make authorization decisions based on that credential. The output of the AU phase, whatever we include, will be made available to later phases in AAA processing, so we can choose to, for instance, pass in the DN that was authenticated. The structure we create can be seen in the Probe either as the output nodeset of our XSLT or by examining the AU debug context variable as shown in Figure 17-13.

[](javascript:PopImage('IMG_414','http://images.books24x7.com/bookimages/id_30903/17fig13_alt.jpg','606','176'))  
Add a note hereFigure 17-13: The final output of our AU stylesheet.

Add a note hereThis XML structure has been generated by DataPower to wrap the output of our stylesheet, and it will be passed to the Map Credentials step.

Add a note hereBecause Web services are inherently stateless, there likely will be multiple inbound requests with the same credentials. For performance reasons, the results of the authentication step, as well as the authorization step, will by default be cached for three seconds. Caching can be just as useful for a custom AAA step, because it will lower the load of any backend systems that the AAA step connects with. However, care should be taken that this caching does not cause undesired effects. For instance, if your custom AU stylesheet connects to a service that maintains an audit log intended to record all authentications, caching would mean that some cached requests would not be recorded, and thus caching should likely be disabled. The cache lifetime is configurable within the definition of the AAA policy, and it can be completely disabled if required.

**Warning: Do Not Rely on the Probe Debug Variables**

Add a note hereFigure 17-13 shows the content of the internal Probe variable var://context/AAA-internal-debug/AU, instead of using the extension trace as previous examples have. This is deliberate in order to raise this important point: Although these variables are extremely useful for debugging, they should *not* be relied upon in your XSLT!

Add a note hereThese variables exist only when the Probe is enabled. This means that, if in your XSLT you read the contents of this or any other of the AAA-internal-debug variables, and make processing decisions based on them, your code will not work when the Probe is disabled.

Add a note hereThe Probe should always be disabled in production environments, which means that testing must be carried out much earlier with the Probe disabled, just to make sure that nothing you are doing is affected by the Probe.

**MC—Map Credentials**

Add a note hereThe Map Credentials (MC) step, as discussed in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926), enables you to modify the output of the authenticate step using a number of methods; one of those methods is to run a custom stylesheet. Of course, when the previous step of authentication was carried out using a stylesheet, in which we were able to present the output of the AU step in any way we chose, there is little value in further modifying it using a second stylesheet in the MC step! Thus, in most instances when AU uses a stylesheet, MC will not.

Add a note hereIf authentication was performed using an out-of-the-box method, however, it can often be beneficial to use a custom stylesheet to perform credential mapping. This is also a nice way of performing further authentication over and above what the out-of-the-box authentication methods provide. For instance, if AU was used to simply bind a DN and password to verify that it is valid (a common use case), the MC step could take these credentials and perform further validation (for instance ensure that the user is not on a specific list of users who should be barred from logging in) or enrichment (such as pulling data out of a database for the specific user which will be used during a later phase such as authorization) or token exchange (for instance via a call out to Tivoli Federated Identity Manager).

Add a note hereOne specific example of a credential mapping step that is often required is to map a distinguished name from x.509 format, as may often be found in a digital certificate, to the format more commonly found in LDAP servers. For example, the distinguished name uid=user1,ou=research,o=example.com would likely be presented in a certificate in x.509 format as o=example.com/ou=research/uid=user1—that is, slashes are used instead of commas and the order is reversed. To use this DN to check against an LDAP server, we would need to map the entries into the reverse order; a stylesheet to perform this mapping is shown in Listing 17-4.

Add a note hereListing 17-4: MC Stylesheet to Map Distinguished Names—MC.xsl

Add a note here<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:dp="http://www.datapower.com/extensions"

xmlns:fn="http://www.w3.org/2005/02/xpath-functions"

xmlns:str="http://exslt.org/strings"

extension-element-prefixes="dp" exclude-result-prefixes="dp"

version="1.0">

<xsl:output method="xml" />

<xsl:template match="/">

<xsl:variable name="originalDN"

select="/credentials/entry[@type='validate-signer']

/CertificateDetails/Subject/text()"/>

<xsl:variable name="splitDN">

<xsl:copy-of select="str:split($originalDN,'/')"/>

</xsl:variable>

<xsl:variable name="newSplitDN">

<xsl:for-each select="$splitDN/token">

<xsl:sort order="descending" data-type="number"

select="position()"/>

<xsl:variable name="this"

select="normalize-space(.)"/>

<xsl:value-of select="concat(',',$this)"/>

</xsl:for-each>

</xsl:variable>

<xsl:variable name="newDN"

select="substring-after($newSplitDN,',')"/>

<entry type='custom'>

<dn><xsl:value-of select="$newDN"/></dn>

</entry>

</xsl:template>

</xsl:stylesheet>

Add a note hereIf the MC stage returns no credentials, the authorization step is likely to fail. Of course, whether or not it does is customizable! The default authorization processes consider authentication to have passed only if credentials are returned by the MC stage; in particular, this is true for the “any authenticated” authorization option, which is perhaps not obvious.

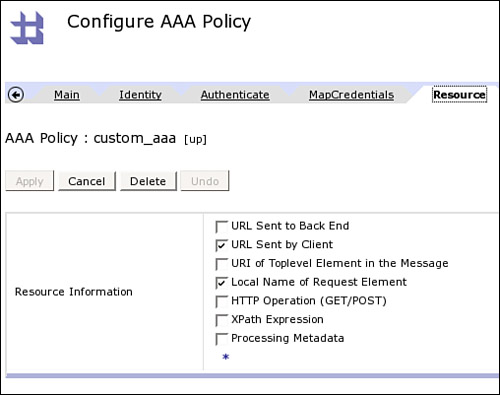
Add a note hereRead that again. The default authorization processes consider authentication to have passed only if credentials are passed to them. What this means is that, if your MC stage changes your authentication, strange and unexpected things can happen. An especially bad case might be if authentication fails, but your credential mapping still outputs a credential; in this case, a user could pass authorization and be allowed to access the protected resource they are trying to, even though authentication failed! Of course, this would likely be due to a bug in your stylesheet, but that is scant comfort when the villains are making off with the swag.

Add a note hereThere may however be good use cases for changing the credentials to be valid after a failed authentication. For example, it is possible that if someone fails authentication, we might still want to allow them access to the system but as a guest user with limited privileges.

**ER—Extract Resources**

Add a note hereThe ER stage is unique among the AAA stages in that it is the only one that cannot be replaced with a custom stylesheet. The output of the ER stage can, however, be modified or even completely removed by using a custom stylesheet in the MR stage which is next.

Add a note hereResource extraction can be flexible. It can include processing metadata, the power of which was shown in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926), an XPath expression to extract any specific piece of the incoming XML message, and a number of other options as shown in Figure 17-14.

[](javascript:PopImage('IMG_415','http://images.books24x7.com/bookimages/id_30903/17fig14_alt.jpg','508','401'))  
Add a note hereFigure 17-14: Resource extraction can be flexible.

Add a note hereA node tree is output by the ER stage that is used as the input to the MR stage that follows. For the example shown in Figure 17-14, with two types of resources extracted, this node tree looks like Figure 17-15.

[](javascript:PopImage('IMG_416','http://images.books24x7.com/bookimages/id_30903/17fig15_alt.jpg','565','142'))  
Add a note hereFigure 17-15: Multiple resources.

**MR—Map Resources**

Add a note hereThe Map Resources stage modifies the extracted resource. Why would we want to do that? The most common use is to add together different resources of different types into a single resource of a specific type that an external authorization system will be able to understand.

Add a note hereLet’s say that our external authorization system, as many do, works by protecting URLs. It can express an authorization constraint in terms of URL pattern matching; for instance, it would be able to protect the following:

Add a note here/bluebank/richpeople/currentaccount

Add a note hereThis happens to be the URI of a Web service served on a backend application server. Our external authorization system can state a constraint that, in order to access this URL, you have to have authenticated. However, as far as the external authorization system is concerned, this is only a URL; it has no concept of Web services.

Add a note hereThis is unfortunate, because under the current account Web service URL shown, our backend application server provides a number of different Web service operations. It has an addMoney operation, a removeMoney operation, and a getAmount operation. What if we want to give one group of users the ability to run just the getAmount operation, and another group of users the ability to addMoney and removeMoney? Or, for instance, what if we require one level of authentication to getAmount, but require extra authentication to addMoney and removeMoney? These are real requirements; authorization at the operation level is a common thing to want to do.

Add a note hereDataPower, of course, is flexible and can easily protect at the Web service operation level. But if the external authorization system cannot understand Web service operations, how can we even ask it who should be allowed to call which operation? That’s where the Map Resources step shines. The stylesheet in Listing 17-5 can perform this mapping.

Add a note hereListing 17-5: MR Stylesheet—MR.xsl

Add a note here<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:dp="http://www.datapower.com/extensions"

extension-element-prefixes="dp" exclude-result-prefixes="dp"

version="1.0">

<xsl:output method="xml" />

<xsl:template match="/">

<!-- take the url and operation name from the ER output -->

<xsl:variable name="url"

select="resource/item[@type='original-url'][1]/text()"/>

<xsl:variable name="opname"

select="resource/item[@type='request-opname'][1]/text()"/>

<xsl:variable name="resource"

select="concat($url,'/',$opname,)"/>

<resource>

<item type="custom">

<xsl:value-of select="$resource"/>

</item>

</resource>

</xsl:template>

</xsl:stylesheet>

Add a note hereThe ‘request-opname’ entry used is the result of selecting Local Name of Request Element in the resource extraction stage. This simple stylesheet concatenates the name of the operation to the URL that the client requested. Now we have a way of expressing the constraint in the external authorization system, by protecting the fictional URLs:

Add a note here/bluebank/richpeople/currentaccount/getAmount

/bluebank/richpeople/currentaccount/addMoney

/bluebank/richpeople/currentaccount/removeMoney

Add a note hereEach of these “URLs” can then be given different authorization constraints in the external authorization system, and the AZ step acts as though the client requested the modified URL.

**AZ—Authorization**

Add a note hereThe authorization step receives a large structure as its input. This node tree contains the output of the EI, MC, and MR steps, such that AZ can make a decision based on whether a given authenticated identity (EI and MC) can access a given resource (MR).

Add a note hereMany out-of-the-box authorization options already exist, and are often a perfect match to customer requirements for authorization. Sometimes, however, there is a need to go beyond these out-of-the-box options, either for integration with a custom external authorization system other than those directly supported, or to expand upon the built-in capabilities. An example of the latter is presented here.

Add a note hereOut-of-the-box integration with LDAP for authorization is implemented as one of the most common use cases—checking for membership of a specific LDAP group. This works by searching the LDAP server for the list of members in that group, and then comparing the output of the MC stage to that list. If the mapped credential is found, authorization succeeds; if not, then it fails. But what if we want to do more than checking for a single group membership? For instance, we might have a number of possible URLs (or Web service methods as demonstrated under MR earlier) and we would like to authorize different groups for each of the URLs, but without having to call out to an external authorization system. This may be an acceptable solution for a small number of static URLs, and would probably be easy to migrate later to a more robust enterprise authorization solution such as TAM.

Add a note hereOne way that we might express such a set of authorization constraints would be to create a file that contains an XML nodeset, where the constraints are stated in a way that they can easily be retrieved using XPath. Listing 17-6 shows such a file.

Add a note hereListing 17-6: One Way in Which Authorization Constraints Could Be Expressed

Add a note here<?xml version="1.0" encoding="UTF-8"?>

<RESOURCES>

<RESOURCE>

<URL>/bluebank/richpeople/currentaccount/getAmount</URL>

<GROUP>cn=workers,ou=groups,ou=example.com</GROUP>

</RESOURCE>

<RESOURCE>

<URL>/bluebank/richpeople/currentaccount/addMoney</URL>

<URL>/bluebank/richpeople/currentaccount/removeMoney</URL>

<GROUP>cn=managers,ou=groups,ou=example.com</GROUP>

</RESOURCE>

</RESOURCES>

</xsl:stylesheet>

Add a note hereThis file would be written to the local file system (in this case to the file named local:///AZresources.xml) and will be called from the stylesheet; it could also be loaded from a remote server.

Add a note hereTo process these constraints, our AU stylesheet must take the inputs from the MR step and know which resource the user is trying to access, take the input from the MC step and discover what groups the user is in, and then compare them to the expressed authorization constraints and make an authorization decision. The input to the AU stylesheet is shown in Figure 17-16.

[](javascript:PopImage('IMG_417','http://images.books24x7.com/bookimages/id_30903/17fig16_alt.jpg','727','387'))  
Add a note hereFigure 17-16: The information from previous steps is available for authorization decisions.

Add a note hereTo further complicate matters, real world LDAP structures are rarely simple and can sometimes require quite complicated interactions! The example in Listing 17-7 demonstrates an example of the kind of interaction with LDAP that is sometimes needed to determine group memberships and uses the authorization constraints stated in Listing 17-6 to make an authorization decision.

Add a note hereListing 17-7: Custom Authorization Example

Add a note here<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:dp="http://www.datapower.com/extensions"

extension-element-prefixes="dp" exclude-result-prefixes="dp"

version="1.0">

<xsl:output method="xml" />

<xsl:template match="/">

<xsl:variable name="perms"

select="document('local:///AZresources.xml')"/>

<xsl:variable name="resource"

select="/container/mapped-resource/resource/item/text()"/>

<xsl:variable name="DN" select="/container/mapped-credentials/

entry[@type='custom']/authenticated/

entry[@type='ldap']/text()"/>

<xsl:variable name="groups" select="dp:ldap-search(

'ldapserver','389','','',$DN,'ibm-allGroups','',

'BASE','','')"/>

<xsl:for-each select="$groups/LDAP-search-results/

result/attribute-value[@name='ibm-allGroups']">

<xsl:variable name="group" select="."/>

<xsl:for-each select="$perms/RESOURCES/

RESOURCE[URL=$resource]/GROUP">

<xsl:if test=".=$group"><approved/></xsl:if>

</xsl:for-each>

</xsl:for-each>

</xsl:template>

</xsl:stylesheet>

Add a note hereThe stylesheet makes an LDAP call to search for a specific attribute of the authenticated user. In this case, we use the IBM Directory Server, and that attribute, ibm-allGroups, is a special attribute of the directory server in use. The attribute returns a list of all the groups that a particular entity is a member of. We then go through the list of groups and search for one that matches any of the group entries in the permissions file for the requested resource. If we find a match, an XML node <approved/> is emitted and authorization succeeds; if not, no output is emitted, which results in an authorization failure. Indeed, any node set apart from <approved> will also be treated as authorization failure; the DataPower runtime is expecting to see an <approved> node.

**Tip: LDAP optimization**

Add a note hereWhile not part of the LDAP specification itself, many LDAP servers have a “special” attribute that returns a list of the groups for a given user. For the IBM Directory Server, this is ibm-allGroups, for Active Directory, it is memberOf, and so on. Using this attribute requests that the directory server itself calculate which groups the user is a member of; this is often significantly faster than going through a list of all of the members of a group and comparing them locally.

Add a note hereThis is because in a typical directory server a given user will be a member of some relatively small number of groups; however, a group may have a very large number of members, so enumerating all of the members for comparison is much more expensive in terms of computation and network bandwidth than asking the directory server to do it for us.

**PP—Post Processing**

Add a note hereFinally, the Post Processing stage comes at the end and enables you to do literally anything based on the output of all the other stages. The post processing input includes

* Add a note hereIdentity
* Add a note hereCredentials
* Add a note hereMapped-credentials
* Add a note hereResource
* Add a note hereMapped-resource
* Add a note hereThe original message

Add a note hereThe default Post Processing options cover many scenarios. DataPower can generate an LTPA token, which is discussed in [Chapter 21](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3908#3908), [“Security Integration with WebSphere Application Server.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3908#3908) It can generate an SAML assertion, a WS-Security UserNameToken, a WS-Security Kerberos token, or a SPNEGO token. It can call out to Tivoli Federated Identity Manager for token mapping. It can run some of the built-in post processing stylesheets such as strip-wssec-header.xsl to strip off WS-Security headers. Or it can execute a custom PP stylesheet to perform any action you want.

**Tip: Credential Trimming**

Add a note hereWhen DataPower is processing incoming credentials and enforcing your security policies by performing authentication, authorization, and audit, it is likely that the credentials will not be required by the systems DataPower is proxying. If that is the case, they should be removed from the request before passing it on. This is best done in the PP stage with a simple custom stylesheet.

## Working with Tivoli Security

Add a note hereTivoli’s Access Manager (TAM) and Federated Identity Manager products (TFIM) are the IBM solutions for externalized authentication and authorization (TAM) and federated identity management (TFIM). These powerful products enable enterprisewide definition of policies for who is allowed to do what in the enterprise, and those policies can be enforced at policy enforcement points, such as DataPower.

Add a note hereBecause the focus of this book is DataPower and not the Tivoli product set, the discussion in this section builds on existing work by the Tivoli DataPower integration team published in the TFIM documentation.

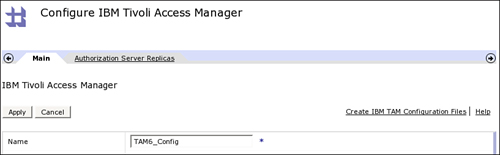
Add a note hereThat documentation shows how to configure integration with the TAM and TFIM products; this chapter explores the configuration of TAM integration.

### Add a note hereIntegration with TAM

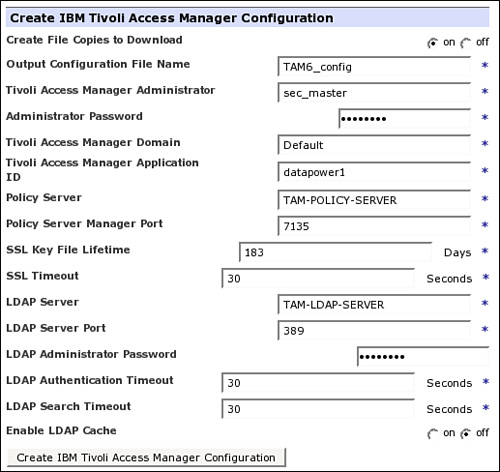
Add a note hereIn order for a DataPower appliance to integrate with TAM, it must have two things: a TAM license on the device and the correct TAM libraries installed in the currently running version of firmware. The firmware contains a built-in configurable TAM client, able to communicate with and externalize authentication and authorization to a specific version of the TAM server. The place to check this is under Status→System→Device Features, where the feature TAM should appear and be set to Enabled, and under Status→System→Library Information, where the library TAM should appear with the correct version. At time of writing, supported TAM versions are version 5 and version 6, and you should use the version of firmware that contains the relevant client version for your TAM server.

#### Initial Configuration

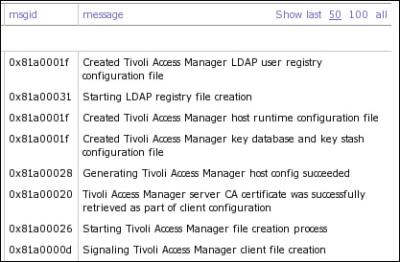
Add a note hereBefore any security integration with TAM can take place, the TAM client must be configured. The configuration process on DataPower works by completing a form to create some configuration files required for TAM integration. This can be found on the TAM configuration object page, which is under Objects→Access→IBM Tivoli Access Manager, as shown in Figure 17-17.

[](javascript:PopImage('IMG_418','http://images.books24x7.com/bookimages/id_30903/17fig17_alt.jpg','817','253'))  
Add a note hereFigure 17-17: From the TAM Object Page, there is a link to create TAM configuration.

Add a note hereClicking the Create IBM TAM Configuration Files link opens up a new window that determines how the files are to be generated. The window is shown in Figure 17-18. Note that in this example we have used static hosts defined on the DataPower device, for the hostname of the TAM Policy Server and of the LDAP Server, although this may use DNS.

[](javascript:PopImage('IMG_419','http://images.books24x7.com/bookimages/id_30903/17fig18_alt.jpg','517','488'))  
Add a note hereFigure 17-18: TAM configuration.

Add a note hereAfter we click the Create IBM Tivoli Access Manager Configuration button shown in Figure 17-18, the appliance begins the file creation process. It makes a call out to the TAM policy server, authenticates to it using the credentials you have supplied, and asks the Policy server to register it as a valid server for sending authentication and authorization requests. As part of this process, TAM creates a new account for the DataPower appliance that it uses to bind to the LDAP server to perform queries as required. It provides the TAM CA certificate to the DataPower appliance such that SSL connections can be authenticated and trusted. Finally all the TAM configuration files are saved to the relevant areas on the DataPower file system. The DataPower logs of this process are shown in Figure 17-19.

[](javascript:PopImage('IMG_420','http://images.books24x7.com/bookimages/id_30903/17fig19.jpg','417','274'))  
Add a note hereFigure 17-19: TAM configuration logs.

Add a note hereThe TAM configuration files that are created consist of four separate files:

1. Add a note hereA text file with a suffix of .conf, which contains specific information obtained from the TAM server about the TAM configuration.
2. Add a note hereA binary file with a suffix of .conf.obf, which contains the same information as the .conf file but in an obfuscated file, and can be used instead of the conf file if there is a need for the passwords used to connect to the LDAP server and TAM to not be stored in plain text.
3. Add a note hereA key database file with a suffix of .kdb, which contains the SSL certificates used to validate the SSL connection with the TAM policy server.
4. Add a note hereA stash file with a suffix of .sth, which contains the stashed password to the key database file.

Add a note hereThe names of the files are determined by the Output Configuration File Name option in the configuration screen; in our example from Figure 17-18, the files would be called TAM6\_config.conf, TAM6\_config.kdb, and so on. Note that the key database file and related stash file are, by default, created only in the cert:// file system, from which they cannot be exported. If you have a need to export these files (for instance for backup purposes), the Create File Copies to Download option should be selected when creating the configuration; they will then be copied to the temporary:// file system from which they can be downloaded. The other two files are stored to the local:// file system of the domain.

##### Application ID

Add a note hereOne of the fields shown in Figure 17-18 is called the “Tivoli Access Manager Application ID.” This field is used to create a server definition for this specific DataPower appliance in the TAM database and a server account in the LDAP server used as its repository.

Add a note hereThe server definition and account that is created uses both the application ID and the System ID of the DataPower appliance. The System Identity of the DataPower device is configured at an appliance level in the Default domain, by navigating to Administration→Device→System Settings and filling in the System Identifier field. Crucially, whatever is put in here must be resolvable on the TAM server, and must resolve to the DataPower appliance, otherwise the configuration will fail. This can be done by modifying the hosts file on the TAM server, or by specifying a resolvable fully qualified DNS name as the System Identifier. In our example, we used the Application ID datapower1 and a System ID of DataPowerBookXI50 (and DataPowerBookXI50 was added to the hosts file of the TAM server), thus the server definition is datapower1-DataPowerBookXI50 and the account principal is datapower1/DataPowerBookXI50.

Add a note hereThe Application ID is used in TAM for application-specific authorization purposes. Thus two DataPower devices that serve the same “application” (for instance which both proxy the same Web service and have a Load Balancer in front of them) should use the same Application ID. Their entries in the TAM namespace differ because of the different System ID’s with which they should be configured.

Add a note hereIf you have a need to configure TAM multiple times on the same appliance, the application ID’s used *must* be different. For instance, you may want to have multiple domains all integrating with TAM for development or testing purposes. Because the System ID is a device-level setting configured in the Default domain, no two DataPower domains on the same device should ever use the same Application ID when connecting to the same TAM server."System Identifier"

**System Identifier**

Add a note hereThe System ID of the DataPower appliance must be set for a TAM configuration to work correctly. If you do not set this on your device, the TAM client will still connect and configure itself, however rather than using the System ID for the creation of the server definition and the account principal, it will use the value entered for the TAM policy server on the configuration screen. Things appear to work, but you have incorrect configuration on the TAM server, and this may cause serious problems later. It is strongly recommended that DataPower appliances using TAM should have correctly configured System IDs.

##### Configuration Failures

Add a note hereIf the TAM configuration fails, this will be clear from the logs in the domain. The failure message likely contain a TAM error; these can be decoded by looking at the TAM documentation, or by asking a friendly TAM administrator. The most likely causes of TAM configuration failure are

* Add a note hereLack of network connectivity. There must be network connectivity between DataPower and the TAM server, not just at runtime but also during configuration. As mentioned, the configuration process involves connecting to and registering with the TAM policy server, and if this is not possible, it will result in a failed configuration.
* Add a note hereIncorrect policy server port. Most TAM policy servers run on the default ports of 7135 and 7136, but this is configurable, and some TAM administrators may have chosen to use a different port.
* Add a note hereIncorrect username or password used for connection to the TAM server or the LDAP server.
* Add a note hereAn account for the specific application ID for a given appliance with a given system identifier already exists in this TAM server, perhaps from a prior configuration attempt.
* Add a note hereIncompatible version of TAM client. For instance using a TAM5 DataPower firmware will not work with a TAM6 Server if that TAM6 server is configured in FIPS mode.

Add a note hereAny issues with the TAM configuration must be resolved before continuing. Do not go on to creation of TAM configuration objects until the TAM configuration has completed successfully and been verified in the logs.

##### Unconfiguration

Add a note hereIf the configuration partially succeeds, or if you need to remove an existing configuration, it is not as simple as just removing the files that are created on the DataPower appliance. Because the configuration process connects to the Access Manager server and makes configuration changes, those must also be removed. This is especially important if you intend to use the same application identity, because the configuration will simply fail.

Add a note hereThe DataPower TAM client does not include a utility to remove the TAM configuration. Thus removing the entries from the TAM database and LDAP server must be done on the server itself. TAM provides a command, svrsslcfg, which is used to manually add and remove server configuration from the database and LDAP server. This command requires a configuration file, an application ID, and a hostname. For removal, it is possible to simply use an empty configuration file; the application id (-n) and the hostname (-h) are the parameters that decide the server definition to be removed. The touch command can be used to create an empty configuration file, which is then passed to the svrsslcfg command. This process is shown in Listing 17-8.

Add a note hereListing 17-8: Unconfiguring a DataPower Appliance from TAM

Add a note heretam-server:~ # pdadmin -a sec\_master

Enter Password: \*\*\*\*\*\*\*\*

pdadmin sec\_master> server list

ivacld-tam-server

datapower1-DataPowerBookXI50

pdadmin sec\_master> quit

tam-server:~ # touch dummy.conf

tam-server:~ # svrsslcfg -unconfig -f dummy.conf -n datapower1 -h

DataPowerBookXI50

Enter the password for sec\_master:

\*\*\*\*\*\*\*\*

Unconfiguration of application "datapower1" for host "DataPowerBookXI50" is

in progress.

This might take several minutes.

SSL unconfiguration for application "datapower1" has completed

successfully.

tam-server:~ # pdadmin -a sec\_master

Enter Password:

pdadmin sec\_master> server list

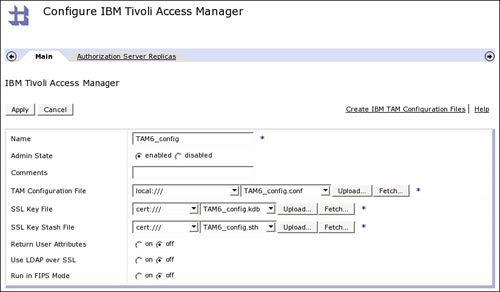
ivacld-tam-server

pdadmin sec\_master> quit

tam-server:~ #

#### Object Creation

Add a note hereAfter the four configuration files are successfully created, we can go ahead and create a TAM configuration object referencing those files, as shown in Figure 17-20.

[](javascript:PopImage('IMG_421','http://images.books24x7.com/bookimages/id_30903/17fig20_alt.jpg','824','482'))  
Add a note hereFigure 17-20: TAM configuration.

Add a note hereOne of the most important fields on Figure 17-20 is the Run in FIPS Mode field. The Federal Information Processing Standard (FIPS) standard specifies explicitly the types of encryption that can be used. If you are working with a TAM server that uses FIPS-compliant encryption, you will be able to communicate with it *only* if this button is selected. FIPS mode is configured in the SSL stanza of the TAM policy server’s pd.conf file—if there is an entry like ssl-enable-fips = yes then the Run in FIPS Mode button must be set to on, otherwise you will see in the logs the error shown in Figure 17-21.

Image from book  
Add a note hereFigure 17-21: Incorrect FIPS settings will result in an SSL error.

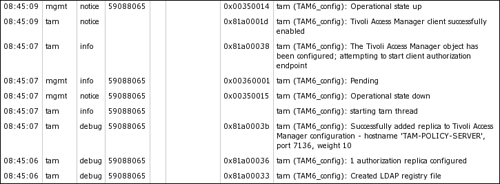
Add a note hereThe second tab of the object screen also needs to be configured with at least one Authorization server replica. (Of course, if there are more we could specify them all here.) In many instances the main authorization server replica resides on the policy server, as it does for our example environment, so we have used the same host alias as before in Figure 17-22. (Note that this is a different port to the manager port shown in Figure 17-18—the default authorization port is 7136, and specifying the wrong port here can be very hard to debug!)

[](javascript:PopImage('IMG_423','http://images.books24x7.com/bookimages/id_30903/17fig22_alt.jpg','823','288'))  
Add a note hereFigure 17-22: At least one Authorization Server replica is required.

Add a note hereAfter clicking Apply on the TAM object configuration screen, we see the object initially go into a red Pending state, shown in Figure 17-23.

[Image from book](javascript:PopImage('IMG_424','http://images.books24x7.com/bookimages/id_30903/17fig23_alt.jpg','793','49'))  
Add a note hereFigure 17-23: Initial Pending state.

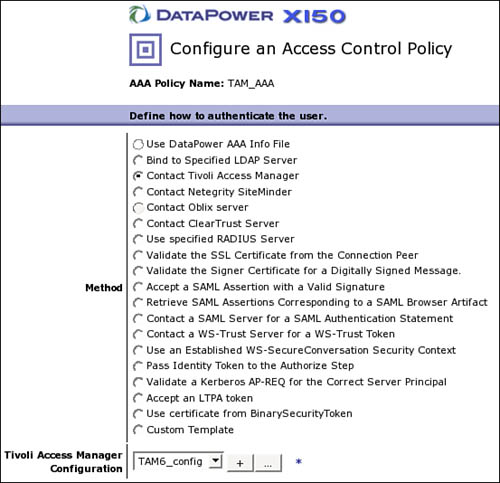
Add a note hereThis is not a cause for alarm—this is simply the state it goes into when the initial configuration is being verified, because it takes a couple of seconds for everything to fall into place. The logs are shown in Figure 17-24.

[](javascript:PopImage('IMG_425','http://images.books24x7.com/bookimages/id_30903/17fig24_alt.jpg','725','267'))  
Add a note hereFigure 17-24: A successful configuration.

Add a note hereThe object creation is now complete and can be used in AAA processing policies.

#### AAA Processing

Add a note hereOnce the TAM configuration is complete and the TAM objects exist, configuring for AAA authentication is as easy as selecting the option and pointing it at the TAM configuration. Figure 17-25 shows the configuration for AAA authentication.

[](javascript:PopImage('IMG_426','http://images.books24x7.com/bookimages/id_30903/17fig25_alt.jpg','553','534'))  
Add a note hereFigure 17-25: AAA Authentication with TAM.

Add a note hereAuthorization, however, is rather more involved. TAM authorization has many aspects, and permissions can be granted at many different levels. Because the TAM runtime is very flexible, there are many possible ways that access control can be configured, and it is beyond the scope of this book to examine them in detail. Instead we will show a specific example of configuration to demonstrate the flexibility and power of the combination of TAM and DataPower.

Add a note hereThe AZ step to communicate with TAM takes three inputs, which can be seen in Figure 17-26, which displays the Probe extension trace of the input into a TAM AZ step.

[](javascript:PopImage('IMG_427','http://images.books24x7.com/bookimages/id_30903/17fig26.jpg','435','148'))  
Add a note hereFigure 17-26: Inputs to the TAM AZ step.

Add a note hereThe inputs are the authenticated username (which in this instance came directly from a TAM AU step, but could of course have come from a custom AU step or an MC step), the resource being requested (which in this case came from the URL sent by the client, but could of course have come from any other ER or MR step), and the action that DataPower believes the client is trying to perform. The resource and the action here require more explanation.

Add a note hereFirst, we will examine the resource. In order to understand how TAM makes authorization decisions, we have to look briefly at the TAM object space. This object space is just a hierarchical listing of objects that can have access control lists (ACL) attached to them. When TAM receives a request for authorization of a resource, that resource is mapped directly to the object space. To demonstrate this, we set up an object space with a very simple resource structure. At the top level is the object /DataPower, and underneath it are three separate objects: object1, object2, and object3. We have chosen this object space to be a simple representation such that we can easily map it to URLs, which is protected by the object space. The URLs to be protected are shown in Listing 17-9.

Add a note hereListing 17-9: The URL Structure to Protect

Add a note herehttp://datapower-ip-address/DataPower/object1

http://datapower-ip-address/DataPower/object2

http://datapower-ip-address/DataPower/object3

Add a note hereListing 17-10 shows the object space of these specific objects in our TAM server and a description of one of the objects.

Add a note hereListing 17-10: TAM Object Space Configured to Protect Our Simple URL Structure

Add a note herepdadmin sec\_master> object list /DataPower

/DataPower/object1

/DataPower/object2

/DataPower/object3

pdadmin sec\_master> object show /DataPower/object1

Name: /DataPower/object1

Description: Test object 1

Type: 14 (Application Container Object)

Is Policy Attachable: Yes

Extended Attributes:

Attached ACL: DataPower-acl1

Attached POP:

Attached AuthzRule:

Effective Extended Attributes:

Effective ACL: DataPower-acl1

Effective POP:

Effective AuthzRule:

pdadmin sec\_master>

Add a note hereThe objects in the TAM namespace have an attached Access Control List (ACL). This is similar to, but far more advanced than, a UNIX© file permission. The ACL attached to our objects is shown in Listing 17-11.

Add a note hereListing 17-11: The ACL Attached to Our Objects

Add a note herepdadmin sec\_master> acl show DataPower-acl1

ACL Name: DataPower-acl1

Description:

Entries:

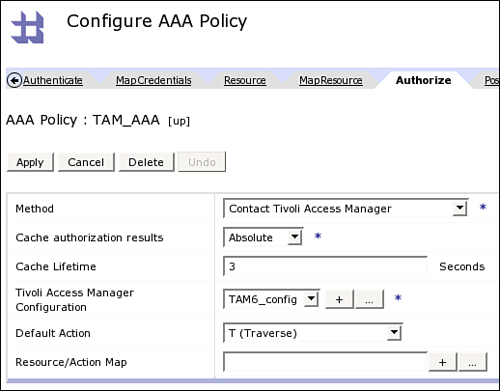
Any-other T

User wasadmin Tr

pdadmin sec\_master>

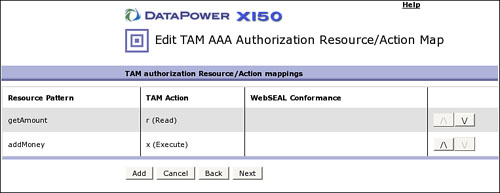
Add a note hereThis ACL is the list of permissions that a specific user has on the object. In this case all authenticated users have the T permission, and the user called wasadmin has both the T and r permissions. T stands for Traverse, while r stands for read. The T permission is required to Traverse the tree; if a user does not have Traverse permission on the object /DataPower, they cannot access anything under it such as /DataPower/object1. The r permission seems like it should mean that the user can read the resource. However, what that actually means is specific to the application that is mapping the resource.

Add a note hereBack in Figure 17-23, we showed the input into the TAM AZ step, which included an authenticated user, a resource, and an action. The action in Figure 17-23 was T; that is, DataPower was asking TAM “Does this user have the T permission on this resource?” The reason it asked about T permission was because this is how the default AZ action is configured. Figure 17-27 shows the TAM AZ configuration.

[](javascript:PopImage('IMG_428','http://images.books24x7.com/bookimages/id_30903/17fig27_alt.jpg','518','405'))  
Add a note hereFigure 17-27: The TAM AZ step.

Add a note hereThe Default action field defines what permission bit we will be asking TAM for when making the authorization call. Because it is set to T, anyone with T permission on the object that matches the resource will be successfully authorized. Our ACL stated that all users have T permission on the object, so anyone who authenticates can access this object. If however we change this to r, only the user named wasadmin, as the only one with r permission listed in the ACL, will be able to access the object.

Add a note hereFinally, we can create a construct called a Resource/Action map, which allows you to dynamically map an operation such as a URL to a required permission bit based on a simple matching, as shown in Figure 17-28.

[](javascript:PopImage('IMG_429','http://images.books24x7.com/bookimages/id_30903/17fig28_alt.jpg','763','294'))  
Add a note hereFigure 17-28: The Resource/Action map.

## Summary

Add a note hereThis chapter has provided an in-depth look at customizing the AAA process, which is extremely flexible and powerful and enables integration with any form of AAA processing imaginable. We have also gone through externalizing of authentication and authorization to IBM’s Tivoli Access Manager, the IBM solution to Enterprise Class centralized Access Control. And yet this knowledge has barely scraped the surface of what is possible with the AAA runtime; you can now go and explore to create the AAA processing that suits your specific requirements and needs!

**Chapter 18: DataPower and SSL**

Add a note hereCryptography 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.

**Add a note here****The Secure Sockets Layer**

Add a note hereSSL 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 tunneling[[1](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=552516796&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch18fn01)] software exists to create an SSL tunnel over which any TCP protocol can be passed.

**Add a note here****Cryptography**

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

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

Add a note hereEach 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**

Add a note hereFor 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.

Add a note hereThe 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*.

Add a note hereSymmetric 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.

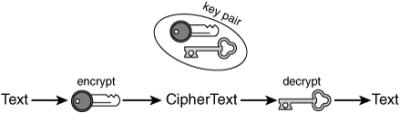
Add a note hereFigure 18-1 shows the encryption and decryption process with a symmetric key; the same key is used to encrypt and then subsequently to decrypt the text.

[Image from book](javascript:PopImage('IMG_430','http://images.books24x7.com/bookimages/id_30903/18fig01.jpg','447','42'))  
Add a note hereFigure 18-1: Encryption using a symmetric key.

Add a note hereAnother 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 way[[2](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=552516796&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch18fn02)] 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*.

Add a note hereKeys 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.

Add a note hereFigure 18-2 shows the encryption/decryption process with a pair of keys.

[](javascript:PopImage('IMG_431','http://images.books24x7.com/bookimages/id_30903/18fig02.jpg','439','125'))  
Add a note hereFigure 18-2: Encryption using an asymmetric key pair.

Add a note hereUsing 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**

Add a note hereCryptography 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).

Add a note hereGiven 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 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.

Add a note hereKeep 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!

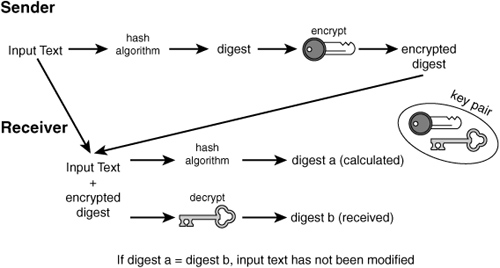
**Impersonation: Digital Signatures**

Add a note hereFinally, cryptography can be used to solve the fundamental problem of impersonation[[3](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=552516796&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch18fn03)] 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.

Add a note hereA 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.

Add a note hereThe 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.

Add a note hereThe flow of message digests is shown in Figure 18-3.

[](javascript:PopImage('IMG_432','http://images.books24x7.com/bookimages/id_30903/18fig03_alt.jpg','568','305'))  
Add a note hereFigure 18-3: Digital signatures use hash algorithms and encryption.

**Digital Certificates**

Add a note hereThe 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.

Add a note hereA 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:

* Add a note hereA copy of the certificate holder’s public key
* Add a note hereA digital signature, signed by a third party (usually *not* the certificate holder)
* Add a note hereInformation about who signed the certificate

Add a note hereThe 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.

Add a note hereThe 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**

Add a note hereThe 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**

Add a note hereAs 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.

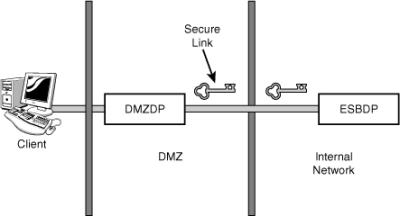
**SSL Authentication and Mutual Authentication**

Add a note hereThe 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.”

**Add a note here****The SSL Handshake**

Add a note hereEnough 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.

Add a note hereThe scenario we use is comprised of two DataPower appliances, one deployed in the De-Militarized 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.

[](javascript:PopImage('IMG_433','http://images.books24x7.com/bookimages/id_30903/18fig04.jpg','443','240'))  
Add a note hereFigure 18-4: Communication between the DMZ and the ESB on the internal network.

Add a note hereThis 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**

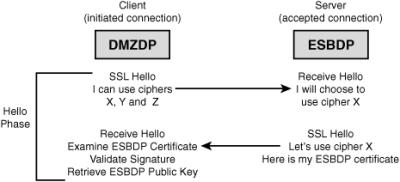
Add a note hereLet’s say that DMZDP initiates the SSL connection to ESBDP, thus making DMZDP the “client” and ESBDP the “server.” DMZDP opens up a TCP socket, and when that is connected, it sends an SSL “hello” message. Inside that hello message is a list of ciphers (hash functions and encryption algorithms)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

Add a note hereESBDP 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.

Add a note hereESBDP’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!

Add a note hereDMZDP 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.

Add a note hereThis interaction is shown more clearly in Figure 18-5.

[](javascript:PopImage('IMG_434','http://images.books24x7.com/bookimages/id_30903/18fig05.jpg','465','212'))  
Add a note hereFigure 18-5: The hello phase.

**Key Exchange**

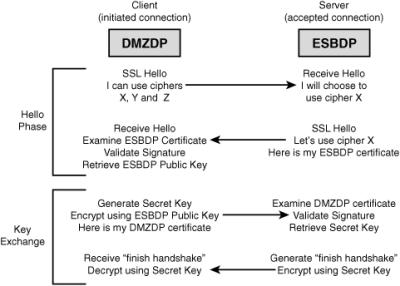
Add a note hereRecall that asymmetric key encryption is relatively expensive, and should be used sparingly. The SSL protocol recognizes this, and only uses it for the initial handshake; all further communication is done using symmetric key encryption. Thus the next stage for DMZDP is to generate a secret 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.

Add a note hereESBDP 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.

Add a note hereAfter 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.

Add a note hereDMZDP 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.

Add a note hereThe full SSL handshake as described is depicted in Figure 18-6.

[](javascript:PopImage('IMG_435','http://images.books24x7.com/bookimages/id_30903/18fig06.jpg','481','344'))  
Add a note hereFigure 18-6: The SSL handshake.

Add a note hereWhew—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**

Add a note hereOne 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.

Add a note hereBy 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**

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

Add a note here[[1](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=552516796&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch18fn01)]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.

Add a note here[[2](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=552516796&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch18fn02)]“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!

Add a note here[[3](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=552516796&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch18fn03)]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.

**Configuring SSL in DataPower**

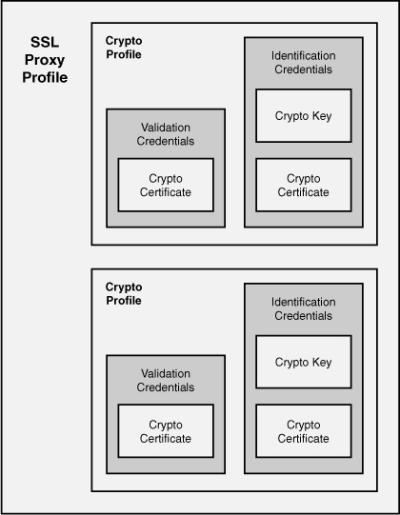
Add a note hereSSL configuration in DataPower often appears complicated at first glance. It requires a number of configuration objects to be created, and the relationship between these objects is sometimes hard to understand. We will examine and define those objects, show where they are used and what 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.

**Add a note here****Configuration Objects**

Add a note hereThis 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.

[](javascript:PopImage('IMG_436','http://images.books24x7.com/bookimages/id_30903/18fig07.jpg','493','387'))  
Add a note hereFigure 18-7: The keys and certificates management page.

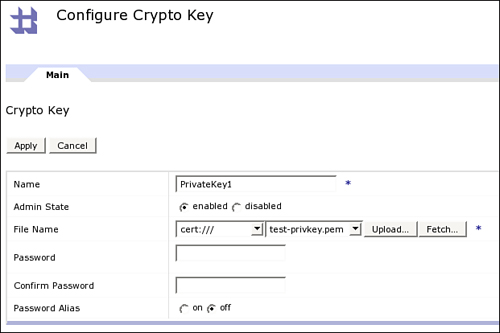
Add a note hereFigure 18-8 contains a diagram of the relationship between the objects required when configuring the use of SSL in DataPower.

[](javascript:PopImage('IMG_437','http://images.books24x7.com/bookimages/id_30903/18fig08.jpg','420','541'))  
Add a note hereFigure 18-8: The relationships between the configuration objects.

Add a note hereEach of these objects is explained in turn.

**The Crypto Key Object—Private Keys**

Add a note hereThe 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.

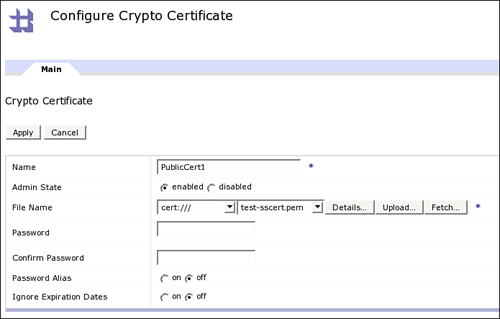
[](javascript:PopImage('IMG_438','http://images.books24x7.com/bookimages/id_30903/18fig09_alt.jpg','621','414'))  
Add a note hereFigure 18-9: A Crypto Key object.

Add a note hereIf 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.

Add a note hereNote 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](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2720#2720), [“Build and Deploy Techniques.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2720#2720)

**The Crypto Certificate Object—Digital Certificates**

Add a note hereJust 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.

[](javascript:PopImage('IMG_439','http://images.books24x7.com/bookimages/id_30903/18fig10_alt.jpg','695','443'))  
Add a note hereFigure 18-10: A Crypto Certificate object.

Add a note hereAgain, 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.

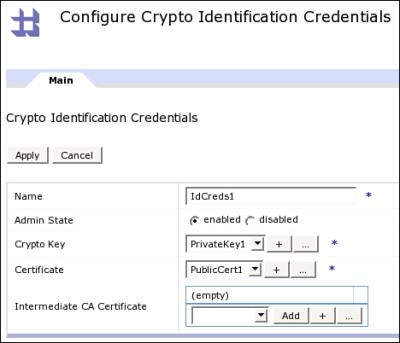
Add a note hereBy default, the Ignore Expiration Date parameter is disabled. In this default state, that means that any object which references this Crypto Certificate object will be marked as down and unusable if the Crypto Certificate object becomes marked down because the certificate it represents 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.

Add a note hereCertificates 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.

Add a note hereThe 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](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3524#3524), [“Web Services Security.”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3595#3595)

**Crypto Identification Credentials—Who Am I?**

Add a note hereThe 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.

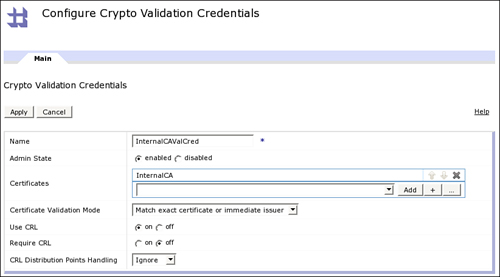
[](javascript:PopImage('IMG_440','http://images.books24x7.com/bookimages/id_30903/18fig11.jpg','468','402'))  
Add a note hereFigure 18-11: A Crypto Identification Credentials object.

Add a note hereA 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.

Add a note hereThe 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?**

Add a note hereA 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.

[](javascript:PopImage('IMG_441','http://images.books24x7.com/bookimages/id_30903/18fig12_alt.jpg','822','455'))  
Add a note hereFigure 18-12: A Crypto Validation Credentials object.

Add a note hereValidation 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).

**Tip: Populating the Validation Credentials**

Add a note hereOne 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.

Add a note hereWell, 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!

Add a note hereIn 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.

Add a note hereThe 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?**

Add a note hereBy 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!

Add a note hereFor 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.

Add a note hereThere 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 validates only 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"

**Certificate Directories**

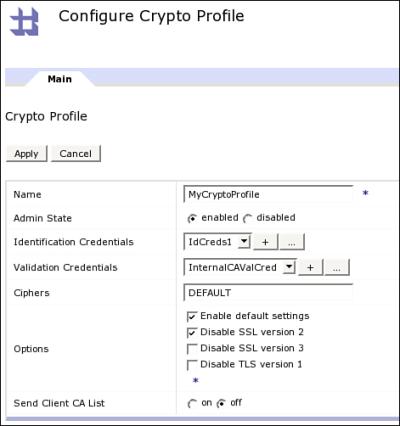
Add a note hereThere are three directories on the DataPower appliance that are commonly used to store certificates. These are the cert:, sharedcert:, and pubcert: directories. Although all three are identical, and in theory certificates can be stored in any of them, each exists to maintain a convention,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

* Add a note here**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.
* Add a note here**Pubcert—** This directory includes the common root certificate authorities that would usually be found in most common Web browsers.
* Add a note here**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.

Add a note hereAlthough 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**

Add a note hereA 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.

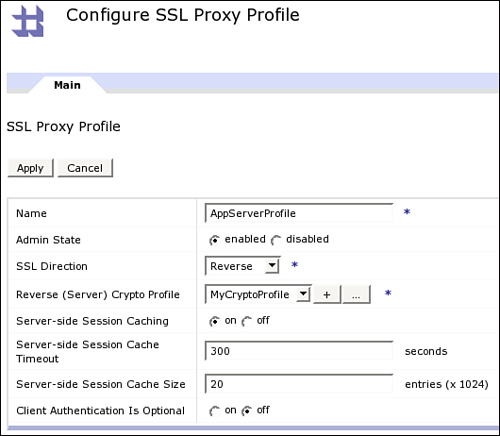
[](javascript:PopImage('IMG_442','http://images.books24x7.com/bookimages/id_30903/18fig13.jpg','471','502'))  
Add a note hereFigure 18-13: A Crypto Profile object.

Add a note hereA 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).

Add a note hereIn 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**

Add a note hereThe 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.

[](javascript:PopImage('IMG_443','http://images.books24x7.com/bookimages/id_30903/18fig14_alt.jpg','530','462'))  
Add a note hereFigure 18-14: An SSL Proxy Profile object.

Add a note hereSSL 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.

* Add a note hereWhen 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.
* Add a note hereWhen 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.
* Add a note hereWhen 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.

Add a note hereThis is really the raison d’etre 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.

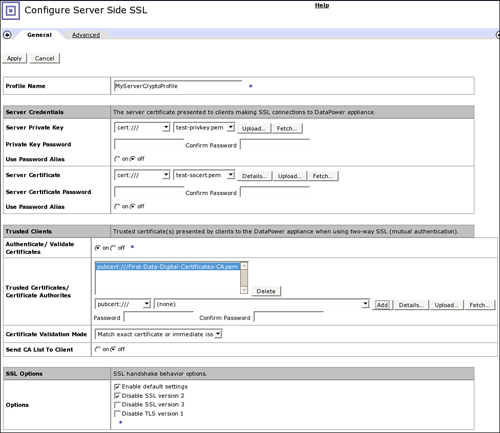
Add a note hereSettings specific to the usage of caching of SSL in this configuration are also set here. These include server and client side SSL caching, which is on by default, and the cache sizes and timeouts for both server and client side.

Add a note hereFinally, the SSL Proxy Profile has a configuration option called “[Client Authentication Is Optional](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3506#3506).” 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.

**Add a note here****Creating Targeted Crypto Profiles**

Add a note hereCrypto 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.

Add a note hereThe 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.

[](javascript:PopImage('IMG_444','http://images.books24x7.com/bookimages/id_30903/18fig15_alt.jpg','976','846'))  
Add a note hereFigure 18-15: The Crypto Profile creation wizard for a Server Crypto Profile.

Add a note hereThe 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.

**Add a note here****SSL Usage Pattern in DataPower**

Add a note hereThe 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.

Add a note hereDue 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.

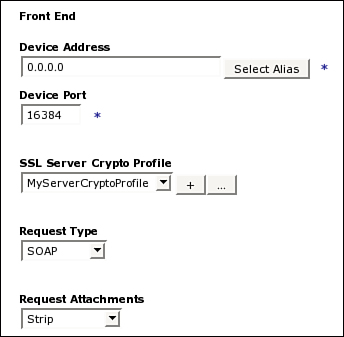
Add a note hereOf course, there is no absolute requirement to use SSL both on the inbound and outbound configuration. There may be valid use cases for a service to use SSL on its inbound connection, but to make a plain unencrypted connection to the backend. (Of course this assumes that you trust everyone 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.

Add a note hereWhichever 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.

**Add a note here****Using SSL—Inbound Configuration**

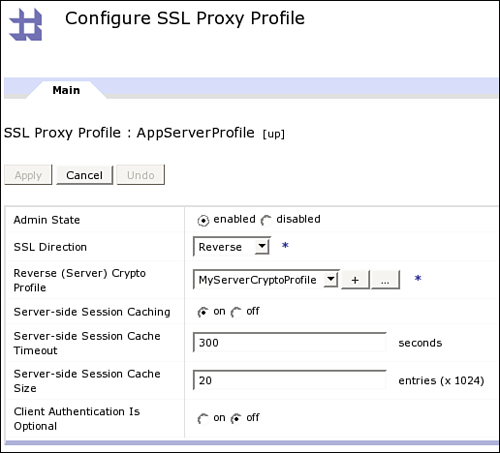
Add a note hereAfter an SSL configuration has been created, it is easy to add that configuration to a service. For instance, a specific Crypto Profile can be reused among multiple XML Firewalls within the same domain, and as part of one or more SSL Proxy Profiles. The same SSL Proxy Profile can be added 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.

Add a note hereTo 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.

  
Add a note hereFigure 18-16: SSL Server Crypto Profile configured on an XML Firewall service.

Add a note hereKeep 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.

Add a note hereThe 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.

[](javascript:PopImage('IMG_446','http://images.books24x7.com/bookimages/id_30903/18fig17_alt.jpg','516','468'))  
Add a note hereFigure 18-17: The same Crypto Profile configured on an SSL Proxy Profile.

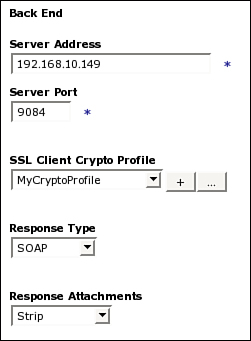
Add a note hereThe FTP Server FSH usage of the proxy profile is more complicated, as described later in this chapter, but the fundamental principle holds true: Assigning an SSL configuration (in this case in the form of an SSL Proxy Profile) to a service is trivial, after all the cryptographic configuration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

**Add a note here****Using SSL—Outbound Configuration**

Add a note hereOutbound 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**

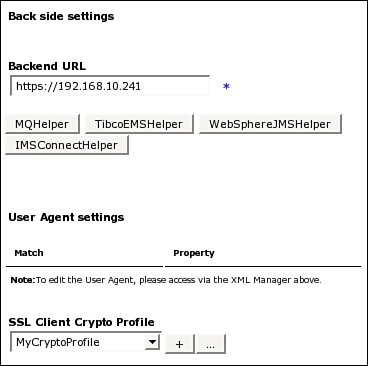
Add a note hereFor simple connections to the backend, a Crypto Profile is specified that contains all the SSL configuration information required. This Crypto Profile is specified slightly differently depending on the type of service. For instance, the XML Firewall uses a Crypto Profile object for connections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

  
Add a note hereFigure 18-18: SSL Client Crypto Profile.

Add a note hereRecall 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!

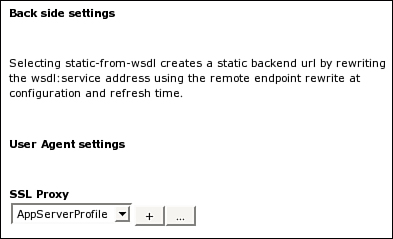
Add a note hereBecause an XML Firewall configuration can use only HTTP or HTTPS and only the server hostname or IP address is specified, this Crypto Profile is used when connecting to the backend server, and the definition of this Crypto Profile results in the call being made using HTTPS instead of plain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

Add a note hereIn 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.

  
Add a note hereFigure 18-19: SSL Client Crypto Profile for the Multi-Protocol Gateway.

Add a note hereHere, 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.)

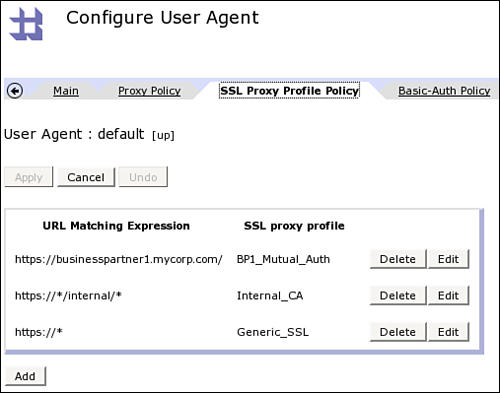
Add a note hereFor the Web Service Proxy, the definition is similarly under “Back side settings,” except that here it uses an SSL Proxy profile, as shown in Figure 18-20.

  
Add a note hereFigure 18-20: SSL Proxy Profile for the back side of a Web Service Proxy.

Add a note hereThe 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**

Add a note hereIn 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.

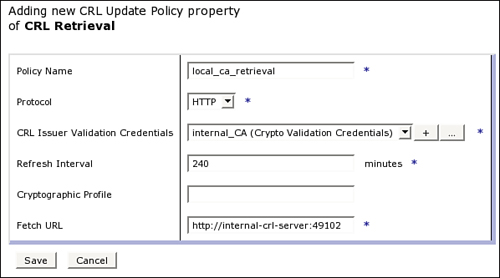
[](javascript:PopImage('IMG_450','http://images.books24x7.com/bookimages/id_30903/18fig21_alt.jpg','507','399'))  
Add a note hereFigure 18-21: SSL Proxy Profile Policy within the User Agent.

Add a note hereFigure 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.

**Add a note here****Certificate Revocation Lists**

Add a note hereOne 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).

Add a note hereA 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.

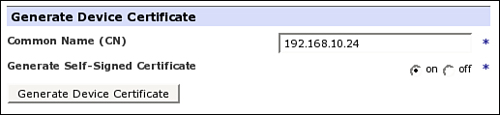
[](javascript:PopImage('IMG_451','http://images.books24x7.com/bookimages/id_30903/18fig22_alt.jpg','598','333'))  
Add a note hereFigure 18-22: CRL retrieval using HTTP.

Add a note hereThe 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.

**Add a note here****Device Certificate**

Add a note hereEach 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!

Add a note hereEvery 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.

[](javascript:PopImage('IMG_452','http://images.books24x7.com/bookimages/id_30903/18fig23_alt.jpg','517','119'))  
Add a note hereFigure 18-23: Generating the Device Certificate.

Add a note hereFigure 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).

Add a note hereIf 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!**

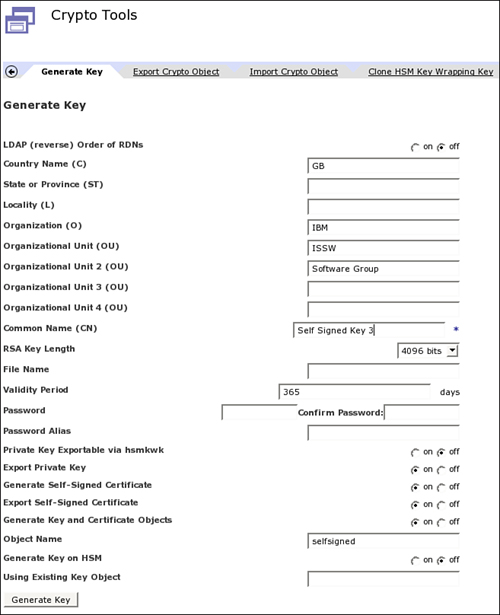
Add a note hereThe two most important things about the device certificate are that you should not use the default certificate that ships on the device itself, and that you should be careful when changing it! If you mistakenly select an SSL proxy profile that is incorrectly configured, it is possible 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**

Add a note hereThe 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.

**Add a note here****Crypto Tools**

Add a note hereThe 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.

[](javascript:PopImage('IMG_453','http://images.books24x7.com/bookimages/id_30903/18fig24_alt.jpg','657','808'))  
Add a note hereFigure 18-24: Crypto tools key generation.

Add a note hereThis 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.

Add a note hereThe 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!**

Add a note hereWhen 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.

Add a note hereWhen 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....

Add a note hereIf 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!

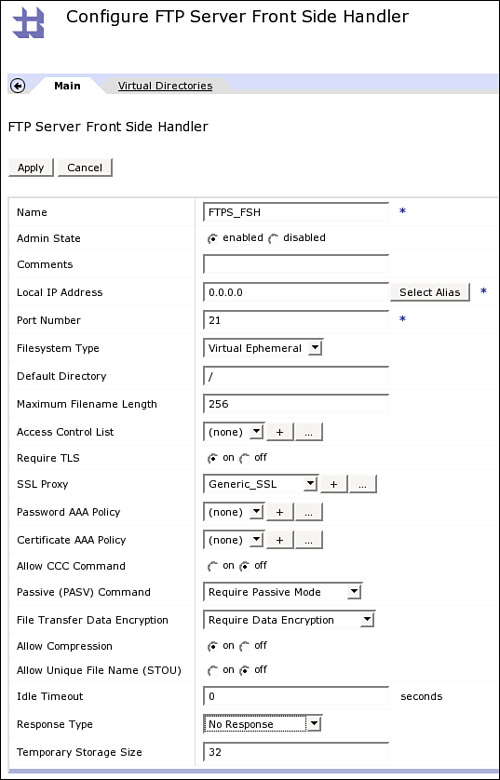
**Add a note here****SSL and FTP**

Add a note hereThe 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.

Add a note hereNote 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.

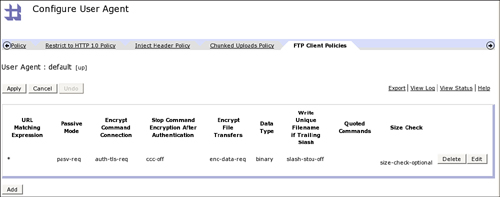
Add a note hereOne of the big differences between FTPS and HTTPS is that SSL with FTPS is explicit. That is, with HTTP you simply specify the URL to be [“https”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3594#3594) and the client will automatically (implicitly) know to negotiate SSL, 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.

Add a note hereFigure 18-25 shows the configuration page for an FTP Server FSH.

[](javascript:PopImage('IMG_454','http://images.books24x7.com/bookimages/id_30903/18fig25_alt.jpg','537','838'))  
Add a note hereFigure 18-25: FTP Server FSH.

Add a note hereOn 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](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=649603881&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ftn.ch18fn04)] If this option is set to on, no FTP 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.

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

[](javascript:PopImage('IMG_455','http://images.books24x7.com/bookimages/id_30903/18fig26_alt.jpg','1016','401'))  
Add a note hereFigure 18-26: FTP client policies.

Add a note hereThe 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.

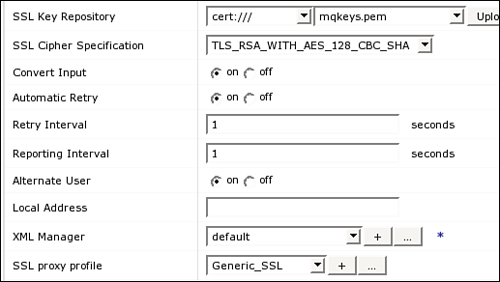
Add a note hereWhen using SSL with FTP, you should always consider two things. First, because the FTP protocol uses multiple sockets for data, whether file listings or files themselves, each data connection requires a separate SSL handshake. Because of this, it is absolutely vital to use both client 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!

Add a note hereSecondly, 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.

**Add a note here****SSL and MQ**

Add a note hereSSL 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.

Add a note hereFigure 18-27 shows part of the definition of an MQ queue manager object.

[](javascript:PopImage('IMG_456','http://images.books24x7.com/bookimages/id_30903/18fig27_alt.jpg','517','292'))  
Add a note hereFigure 18-27: An MQ Queue Manager object.

Add a note hereThis 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.

Add a note hereIf 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.

Add a note hereThe 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.

Add a note hereWhether 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.

| Add a note hereTable 18-1: Mapping of MQ SSL Ciphers to DataPower Crypto Profile Options  [[http://www.books24x7.com/images/b24-bluearrow.gif](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=649603881&objectid=ch18table01&objecttype=spreadsheet)Open table as spreadsheet](http://www.books24x7.com/outputobject.asp?bookid=30903&chunkid=649603881&objectid=ch18table01&objecttype=spreadsheet) | | |
| --- | --- | --- |
| **Add a note hereMQ SSL Cipher** | **Add a note hereSSL Cipher Specification** | **Add a note hereSSL Options** |
| Add a note hereNULL\_MD5 | Add a note hereNULL-MD5 | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereNULL\_SHA | Add a note hereNULL-SHA | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereRC4\_MD5\_EXPORT | Add a note hereEXP-RC4-MD5 | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereRC4\_MD5\_US | Add a note hereRC4-MD5 | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereRC4\_SHA\_US | Add a note hereRC4-SHA | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereRC2\_MD5\_EXPORT | Add a note hereEXP-RC2-CBC-MD5 | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereDES\_SHA\_EXPORT | Add a note hereDES-CBC-SHA | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereRC4\_56\_SHA\_EXPORT1024 | Add a note hereEXP1024-RC4-SHA | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereDES\_SHA\_EXPORT1024 | Add a note hereEXP1024-DES-CBC-SHA | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereTRIPLE\_DES\_SHA\_US | Add a note hereDES-CBC3-SHA | Add a note hereOpenSSL-default+Disable-TLSv1 |
| Add a note hereTLS\_RSA\_WITH\_AES\_128\_CBC\_SHA | Add a note hereAES128-SHA | Add a note hereOpenSSL-default |
| Add a note hereTLS\_RSA\_WITH\_AES\_256\_CBC\_SHA | Add a note hereAES256-SHA | Add a note hereOpenSSL-default |
| Add a note hereAES\_SHA\_US | Add a note hereAES128-SHA | Add a note hereOpenSSL-default |

**Add a note here****When Signing Isn’t Enough**

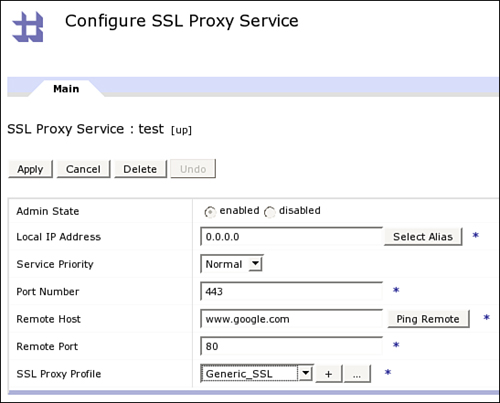
Add a note hereRecall that with SSL, trust is established by checking whether the digital signature of a certificate is signed by a trusted third party that we have chosen to trust. Under DataPower we configure who we trust by creating a validation credential object.

Add a note hereWhat 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.

Add a note hereAs described in [Chapter 16](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926), [“AAA,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=2926#2926) 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](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3157#3157), [“Advanced AAA,”](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3157#3157) to make our identity extraction use a custom stylesheet and examine any of the signed data that we so choose.

**Add a note here****The SSL Proxy Service**

Add a note hereWhat’s that? Haven’t we already talked about SSL Proxies? Well, no—earlier we discussed SSL Proxy *Profiles*, but the SSL Proxy [*Service*](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=5268#5268) is a completely separate kind of service that deserves mention. Figure 18-28 shows an SSL Proxy Service configuration.

[](javascript:PopImage('IMG_457','http://images.books24x7.com/bookimages/id_30903/18fig28_alt.jpg','547','441'))  
Add a note hereFigure 18-28: The SSL Proxy Service.

Add a note hereThe 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.

Add a note hereA 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.

Add a note hereThe two use-cases are therefore

1. Add a note hereTo proxy an inbound SSL connection to an outbound TCP connection, allowing transparent addition of SSL capabilities to a non-SSL service
2. Add a note hereTo 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

Add a note hereUse 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.

Add a note hereTo 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.

Add a note hereUse 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.

Add a note hereTo 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.

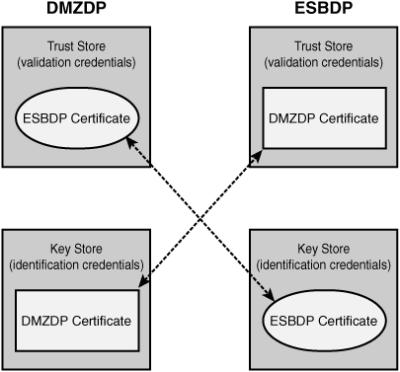
**Add a note here****The Mutually Authenticated Self-Signed SSL Tunnel**

Add a note hereFinally, we examine and describe a very important usage pattern that is an extension of that described earlier in the “[The SSL Handshake](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3349#3349)” section.

Add a note hereThe 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.

Add a note hereThe 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!

Add a note hereFigure 18-29 shows the configuration used here.

[](javascript:PopImage('IMG_458','http://images.books24x7.com/bookimages/id_30903/18fig29.jpg','405','377'))  
Add a note hereFigure 18-29: The mutually authenticated self-signed SSL tunnel.

Add a note hereAs you can see from Figure 18-29, the validation credentials for DMZDP contain exactly one certificate, which is the self-signed certificate of ESBDP, and vice versa—the validation credentials for ESBDP 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 ESBDP has ESBDP’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.

Add a note hereThis 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**

Add a note herePeople 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!

Add a note hereOf 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](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3524#3524).

Add a note here[[4](http://www.books24x7.com/assetviewer.aspx?bookid=30903&chunkid=649603881&noteMenuToggle=0&hitSectionMenuToggle=0&leftMenuState=1" \l "ch18fn04)]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.

**Troubleshooting SSL**

Add a note hereAs 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?

**Add a note here****What Can Go Wrong?**

Add a note hereSSL 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**

Add a note hereThe 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**

Add a note hereRecall 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.

Add a note hereThis 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**

Add a note hereRecall 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).

**Add a note here****Debugging SSL**

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

**Debug Logs**

Add a note hereBecause 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**

Add a note hereIf 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**

Add a note hereMany 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.

Add a note hereAn 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.](http://www.ibm.com)

Add a note hereListing 18-1: Curl Provides Detailed SSL Debugging Information

Add a note here[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]$

Add a note hereNote 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

Add a note hereIn 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.

Add a note hereThe general cryptographic concepts presented will serve as a base for the equally valuable information in the [next chapter](http://www.books24x7.com/assetviewer.aspx?bkid=30903&destid=3524#3524), which discusses Web Services security.