How Secure are your Channels?

By Morag Hughson

So, you've gone to great lengths to control who has access to your queues, but would you care if someone could see the contents of your messages as they were transported across the network to another queue manager? Have you thought about encrypting the traffic across your channels? Would you like to be able to automatically authenticate the partner queue manager? Would you like all this to be integrated with your channels and provided free with WebSphere MQ?

In the new release of WebSphere MQ (formerly MQSeries) V5.3, a protocol known as Secure Sockets Layer (SSL) has been introduced into MQ channels. It is a protocol widely used in many products which transport information across insecure networks; you are probably already using it with your web browser, for example. The SSL protocol provides us with the security benefits of partner authentication, encryption and message integrity.

Building Blocks

The SSL protocol relies on several different technologies combined to provide these benefits. These building blocks are symmetric and asymmetric encryption techniques and hash functions and are described below.

Symmetric Encryption

Symmetric or shared key encryption is a relatively fast encryption mechanism but has problems which stem from the need to keep the key completely secret. Only the partners exchanging information can know the key, or the encryption can be broken.

Asymmetric Encryption

Asymmetric or public/private key encryption, by comparison to symmetric key encryption, is not as fast, but has the benefit that the public key portion of the public/private key pair can be published to the world. The encryption algorithm used is based upon a mathematical function known as a trapdoor function. A trapdoor function is one that cannot be reversed without the use of brute force (i.e. attempting every possible combination until the correct result is discovered). Asymmetric encryption can be done using the public key, by anyone, ensuring that only the owner of the private key can decrypt the message. Asymmetric encryption can also be done using the private key, by the owner of that key, ensuring that anyone can decrypt the message using the public key. This means that the message must have originated from the owner of the private key.

Hash Function

A hash function is a way of producing a small fixed sized number that is a representation of the message being delivered. If the message is changed in any way, the likelihood that the value produced by the hash function is the same is very improbable. This provides a means to detect tampering. The small fixed sized number is known as the hash value, the message authentication code (MAC), or the message digest.

Combining these Building blocks

These building blocks alone are not enough to provide a secure protocol, but it is the manner in which they are combined which provided the security benefits that we want.

Digital signature

A hash function allows detection of tampering. However, someone tampering with a message could also tamper with the hash value so that it matched the new message. By encrypting the hash value using a private key, the sender creates what is known as a digital signature. The receiver checks that the digital signature is valid by decrypting it using the corresponding public key pair to obtain the sender's hash value. This value is then compared to a hash value for the message received, which has been calculated by the receiver. If the values match, the message has not been tampered with and the origin of the message can be trusted.

Certificate

The use of public/private key pairs is clearly a very important part of the SSL protocol. However, how can you be sure you trust the public key you are using. You may have been sent a public key to use, but can you be sure you have the correct key? In a 'man-in-the-middle' attack, the public key you have been sent is intercepted and replaced with the public key of the attacker. The solution to this problem is the use of items known as certificates. They are issued by a well known trustworthy third party known as a Certification Authority (CA) and contain the identity of the owner of the public/private key pair and the public key from that pair. As certificates are delivered across an insecure network, they can be thought of as just another message that could be tampered with. In order to protect against tampering of certificates, they are also signed with a digital signature made from the CA's private key which can be decrypted by anyone using the CA's public key in order to ensure the validity of the certificate.

The identity in a certificate is stored in a particular format known as a Distinguished Name (DN). A DN contains several sub fields allowing the identity, job description and address of the entity to be specified.

Authentication

Partner authentication is achieved by both partners exchanging certificates and validating that these certificates have been correctly signed by the CA and can therefore be trusted. Optionally, the responder can choose to allow anonymous connections, but the responder will always be authenticated. In other words you can have anonymous clients, but you cannot have anonymous servers.

Encryption and Message Integrity

There are a variety of different symmetric key encryption algorithms to use, and also several hash functions. The combination of these two algorithms is known as a CipherSpec and is negotiated as part of the handshake that creates a secure session using SSL. The problem of keeping the symmetric key secret between the two partners is solved in SSL by using asymmetric encryption to deliver the shared secret key, which is then used for all subsequent traffic on that session.

SSL in WebSphere MQ channels

Certificates

To make use of SSL, you must acquire a certificate for each party that wishes to communicate securely. In the case of WebSphere MQ, this means one certificate for each queue manager when using MCA channels, and also one certificate for each logged-on user ID when using MQI channels. The location of this certificate varies from platform to platform. It can be specified as a queue manager's certificate by using the ALTER QMGR command to indicate the location, or can be specified for use with a client application by either using an environment variable or through MQCONNX. The label of the certificate is also important and indicates which queue manager or logged-on user ID owns the certificate. The label is the queue manager name or logged-on user ID appended to the end of `ibmWebSphereMQ`. This string is all in lower case on Unix platforms.

CipherSpecs

A particular encryption algorithm and hash function is known as a CipherSpec. CipherSpecs are specified on a per-channel basis using a parameter on the channel definition called SSLCipherSpec (SSLCIPH). For a secure channel to start up, both ends must specify the same CipherSpec.

Partner Authentication

To specify whether this responding channel is willing to accept anonymous connections, you use a parameter on the channel definition called SSLClientAuthentication (SSLCAUTH) which can be set to one of two values; REQUIRED or OPTIONAL.

Identity Filtering

You may wish to ensure that only certain queue managers can connect to a particular channel. You can filter out unwanted entities by means of their DN. To specify the DN's that are allowed to connect, you use a parameter on the channel definition called SSLPeerName (SSLPEER) which can contain a DN filter which is compared to the DN in the partners certificate.

By Example

In the following example, we have a z/OS queue manager (MQ46) and an AIX queue manager (MORAGAIX) and we are going to set up secure channels between them. Instead of using a CA to sign the certificates for this example, we are going to use self-signed certificates. The only way to authenticate a self-signed certificate is to have a copy of it. So in each case, after the certificate has been created, we will ftp it to the other machine. This example has been tested as it appears here apart from the connection names which have been changed to make it clear what address needs to be used. Be sure to be accurate with the case of the certificate labels.

Certificate Creation

We will show examples of the RACF commands needed to create a self-signed certificate on z/OS and how to use the iKeyMan tool to create a self-signed certificate on AIX. If you want to try this out on other platforms there are specific details in the *WebSphere MQ Security* manual.

Certificate creation on Z/OS

Create a certificate in RACF using the following command (where the user ID we're using (HUGHSON) must be the channel initiator user ID):

```
RACDCERT ID(HUGHSON) GENCERT
SUBJECTSDN(CN('Morag Hughson')
           T('Software Engineer')
           OU('WebSphere MQ Devt')
           O('IBM')
           L('Hursley')
           SP('Hampshire')
           C('UK'))
WITHLABEL('ibmWebSphereMQMQ46')
```

Export the certificate to a dataset so that it can be ftp-ed it to the AIX box, using:

```
RACDCERT ID(HUGHSON)
EXPORT(LABEL('ibmWebSphereMQMQ46'))
DSN('HUGHSON.ZOS.CERT') FORMAT(CERTB64)
```

Create a keyring in RACF using the following command:

```
RACDCERT ID(HUGHSON) ADDRING(MQ46RING)
```

Connect the certificate you created to the keyring using the following command:

```
RACDCERT ID(HUGHSON) CONNECT(ID(HUGHSON)
LABEL('ibmWebSphereMQMQ46') RING(MQ46RING)
USAGE(PERSONAL))
```

Once you have ftp-ed your AIX queue manager's certificate to z/OS, import it into to RACF using the following command (you may want to come back to here later):

```
RACDCERT ID(HUGHSON) ADD('HUGHSON.AIX.CERT')
WITHLABEL('ibmWebSphereMQMORAGAIX')
```

and connect it to your keyring using the following command:

```
RACDCERT ID(HUGHSON) CONNECT(ID(HUGHSON)
LABEL('ibmWebSphereMQMORAGAIX') RING(MQ46RING)
USAGE(PERSONAL))
```

Check that your key ring looks as it should with the following command:

```
RACDCERT ID(HUGHSON) LISTRING(MQ46RING)
```

which gives the following output:

```
Digital ring information for user HUGHSON:

   Ring:
        >MQ46RING<
   Certificate Label Name           Cert Owner     USAGE      DEFAULT
   --------------------------------  ------------   --------   -------
   ibmWebSphereMQMQ46               ID(HUGHSON)    PERSONAL     NO
   ibmWebSphereMQMORAGAIX           ID(HUGHSON)    PERSONAL     NO
```

Certificate creation on AIX

Before starting the iKeyMan tool you must set an environment variable:

```
export JAVA_HOME=/usr/mqm/ssl/jre
```

Then start the iKeyMan tool using the following command:

```
gsk6ikm
```

Make a new **Key Database File** of type CMS with a file name of `key.kdb` and a location of `/var/mqm/qmgrs/MORAGAIX/ssl` and when prompted for a password you must make sure you check the box to stash it to a file.

Now **Create** a **New Self-Signed Certificate** with a key label of `ibmwebspheremqmoragaix`. Take special care to have the key label all in lower case. Use a version of **X509 V3**, fill in Common Name and Organization at the very least and select your Country.

Now extract the certificate you have just created to a file (I used `cert.arm`) so that it can be ftp-ed to your z/OS system. I used a data type of **Base64 encoded ASCII data**.

Now ftp to your z/OS system and put your AIX certificate there and retrieve your z/OS certificate. Use the following ftp settings:

```
ftp> ascii
ftp> quote site recfm=vb
ftp> put cert.arm AIX.CERT
ftp> get ZOS.CERT zoscert.arm
```

Select 'Signer Certificates' instead of 'Personal Certificates' and **Add...** your z/OS certificate file giving it a label.

Ensure the `mqm` group has read permissions on the `key.kdb` and `key.sth` files you have just created.

Remember to go back to the z/OS certificate management steps and import the AIX certificate into RACF.

MQ definitions

We will show examples of MQSC commands to set up the MQ definitions on a z/OS queue manager and on an AIX queue manager.

MQ Definitions on z/OS

Specify the location of the queue manager’s certificate and define some necessary TCBs for use in the channel initiator address space.

```
ALTER QMGR SSLKEYR(MQ46RING) SSLTASKS(5)
```

Add *sslhlq*.SGSKLOAD to your CHINIT STEPLIB and start the channel initiator and a listener.

```
START CHINIT
```

```
START LSTR TRPTYPE(TCP) PORT(1546)
```

Define a sender channel to the AIX queue manager.

```
DEFINE CHANNEL(TO.MORAGAIX) CHLTYPE(SDR)
TRPTYPE(TCP) CONNAME(‘aix-mach(1414)’)
XMITQ(MORAGAIX) SSLCIPH(RC4_MD5_US)
```

```
DEFINE QLOCAL(MORAGAIX) USAGE(XMITQ)
```

Define a receiver channel.

```
DEFINE CHANNEL(TO.MQ46) CHLTYPE(RCVR) TRPTYPE(TCP)
SSLCIPH(DES_SHA_EXPORT) SSLPEER(‘OU=”WebSphere MQ
Devt”’)
```

MQ definitions on AIX

Enter the following MQSC commands using:

```
runmqsc MORAGAIX
```

We have used the default location of the key database file. You can check it using:

```
     1 : display qmgr sslkeyr
AMQ8408: Display Queue Manager details.
   SSLKEYR(/var/mqm/qmgrs/MORAGAIX/ssl/key)
   QMNAME(MORAGAIX)
```

Define a sender channel to the z/OS queue manager.

```
DEFINE CHANNEL(TO.MQ46) CHLTYPE(SDR) TRPTYPE(TCP)
CONNAME(‘mvs-mach(1546)’) XMITQ(MQ46)
SSLCIPH(DES_SHA_EXPORT)
```

```
DEFINE QLOCAL(MQ46) USAGE(XMITQ)
```

Define a receiver channel

```
DEFINE CHANNEL(TO.MORAGAIX) CHLTYPE(RCVR)
TRPTYPE(TCP) SSLCIPH(RC4_MD5_US)
SSLPEER('OU="WebSphere MQ Devt"')
```

Remember to start your listener.

```
runmqlsr -m MORAGAIX -t tcp -p 1414
```

Start your secure channels

Once you have created all these definitions and have remembered to ftp your self-signed certificate to the other machine and add it to the relevant key repository, you are now ready to start your secure channel.

Start your secure channel from the z/OS queue manager to the AIX queue manager and display its channel status once started.

```
START CHANNEL(TO.MORAGAIX)
```

```
DISPLAY CHSTATUS(TO.MORAGAIX) RQMNAME SSLPEER
```

```
CSQM201I !MQ46 CSQXDMSG  DIS CHSTATUS DETAILS 518
CHSTATUS(TO.MORAGAIX)
CHLDISP(PRIVATE)
XMITQ(MORAGAIX)
CONNAME(xxx.xxx.xxx.xxx)
CURRENT
STATUS(RUNNING)
STOPREQ(NO)
RQMNAME(MORAGAIX)
SSLPEER(CN=Morag Hughson;OU=WebSphere MQ Devt;O=IBM;L=Hursley;SP=Hamps
hire;C=GB;)
CSQXDMSG END CHSTATUS DETAILS
```

Start your secure channel from the AIX queue manager to the z/OS queue manager and display its channel status once started.

```
START CHANNEL(TO.MQ46)
```

```
     3 : dis chs(TO.MQ46) RQMNAME SSLPEER
AMQ8417: Display Channel Status details.
   CHANNEL(TO.MQ46)                        XMITQ(MQ46)
   CONNAME(winmvs41(1546))                 CURRENT
   CHLTYPE(SDR)                            STATUS(RUNNING)
   SSLPEER(CN=Morag Hughson,T=Software Engineer,OU=WebSphere MQ
Devt,O=IBM,L=Hursley,ST=Hampshire,C=UK)
   RQMNAME(MQ46)
```

Note that in the channel status, you can see the DN from the certificate of the partner so you can see exactly who is connected to your channel.

Conclusion

WebSphere MQ V5.3 provides a mechanism to secure your channels that is integrated into the product. It does not involve any changes to your MQ applications in order to use it. It is set up as part of your channel configuration and provides a built in mechanism to prevent eavesdropping, tampering and impersonation using a well known and extensively tested security protocol.

What is described here is a basic introduction to the new feature and more detail can be found in the *WebSphere MQ Security* manual that is part of the WebSphere MQ V5.3 product.