IBM Techdoc: 7048145

MQ and SSL/TLS Demystified

Part 1:

Troubleshooting MQ Certificate Issues

Date last updated: May 19, 2016

Mike Cregger – mike_cregger@us.ibm.com
IBM MQ Support

+++ Objective

The objective of this technical document is to provide information on Troubleshooting MQ SSL/TLS issues. In this Part 1, we will deal specifically with SSL Keystore and certificate issues.

Table of Contents

Overview: 3
Part 1 - Troubleshooting MQ certificate issues: 4
Some MQ SSL Basics: 4
SSL Server/Client: 5
A simplistic view of a certificate: 7
Basic MQ Management commands: 8
Simplified MQ certificate process: 9
Troubleshooting Keystore/Certificate issues: 10
I - Certificate keystore exists and is valid/accessible 10
II - Certificates exist, Certificate names are correct 11
III - Certificate chain to a CA Root certificate exists and is valid 12
IV - Certificates are marked as "Trusted" 13
V - Certificate dates are good 13
VI - Certificates are not revoked 14
VII - Remote certificate passed during SSL negotiation is validated 15
VIII – Ensure the correct signer certificates were exchanged 21
Diagnostics to collect if IBM support needed: (for SSL keystore/certificate issue) 23
Related Links and Information: 24

Overview:

MQ and SSL /TLS demystified: Part 1 - Troubleshooting MQ Certificate Issues:
For simplicity in the document, SSL refers to both SSL and TLS protocols.

To use SSL in MQ, there are 2 main things that need to be accomplished/correctly configured.

1) certificate and keystores configured correctly (part 1)
2) cipherspecs / ciphersuites set correctly (part 2)

Once these 2 are correct, SSL is like a black box, it usually just works.
The hard part is getting these 2 correct, especially since we don't make changes here too often, except if certificates are expiring or configuring new connections.

(I just had a critical issue where a certificate expired. The customer decided to request a new one and go to SHA-2 certs, new cert, newer signer certs...Getting the keystores updated correctly on both sides can be tricky.)

Each of these can be complicated and confusing, especially if it was configured a long time ago and just working but now you need to make some changes...

There are other configuration options that can cause issues, like SSLPEER settings and compatibility issues between certificates and ciphers.

If you are having issues, I suggest you first get the basic SSL communications working before configuring additional options.

Additionally, there have been changes in different versions of MQ, cipherspecs supported, SHA-2 support, etc See the links at the bottom for support and resolutions to some issues with these. (Leaving ciphers for Part 2)

So .. on to Part 1 - Troubleshooting MQ certificate issues

Part 1 - Troubleshooting MQ certificate issues:

Most issues are configuration issues. Usually these issues are due to missing certificates or not having the correct certificates.

Common error messages that indicate issues with the certificates:
(Many times these are somewhat vague, and can indicate an issue at either side of the connection.)

AMQ9633: Bad SSL certificate for channel '<insert one>'.
AMQ9637: Channel is lacking a certificate.
AMQ9642: No SSL or TLS certificate for channel '<insert one>'.
AMQ9660: SSL key repository: password stash file absent or unusable.
AMQ9663: An invalid SSL certificate was received from the remote system.
2393: MQRC_SSL_INITIALIZATION_ERROR
2397: MQRC_JSSE_ERROR

Some MQ SSL Basics:

For MQ channel communications to use SSL, there are a couple requirements.
SSL communications involves:

- SSL ciphers - used to communicate over the wire
- Certificates - used to identify and validate each of the entities communicating.

1) For an MQ queue manager to do SSL communications, the queue manager MUST have a keystore, and in most cases it also needs a personal certificate. (ie: queue manager personal certificate required if SSL is configured on Receiver or SVRCONN channels, in effect the SSL server side of a connection)

- An MQ queue manager uses a CMS type keystore to hold its personal certificate(s) and also signer certificates and certificate requests.
- The keystore location and base file name is identified in the queue manager's SSLKEYR property.

2) If a Channel has the SSLCIPH property set, then SSL communications is attempted. If there is any issue/error during SSL initialization/negotiation, the channel will fail to start.

- Both sides of the communication MUST be configured for the same cipherspec/ciphersuite.. Ciphersuites are used when dealing with Java. We will deal with issues/troubleshooting this in Part 2.
- Both sides MUST have the needed certificate keystores/truststores and certificates (personal and signer certs).

3) As part of the SSL initialization, certificates are used to identify the connecting users and

also to validate they are "trusted" and valid.
Client applications also have keystores which contain certificates.

- Java/JMS based MQ client applications use JKS (Java KeyStore files) to hold certificates;
- whereas other MQ client applications (c-based and other) use CMS type keystores.

Certificates get validated in several ways,

- certificate chains are checked,
- certificates revocation checks are done (OCSP/CRL),
- certificate dates are checked, etc..
- trust value
- personal certificates are checked on both sides of the connection, initially on the SSL server side, then information from the personal certificate is also validated on the remote side (SSL client side). ie: ensure the remote keystore contains the exchanged signer certificates, SSLPEER may be checked, etc..

SSL Server/Client:

SSL certificate management concepts for MQ are similar to any other products which do SSL communications.

SSL involves communications between 2 endpoints,
an SSL Server and an SSL Client.

Examples:

MQ client to MQ Queue manager
(SSL Client) to (SSL Server)

MQ Queue Manager Sender channel to MQ Queue Manager Receiver channel
(SSL Client) to (SSL Server)

If the MQ channel is configured for SSL (based on the SSLCIPH property), the SSL/TLS handshake is done as one of the first things on an MQ channel.

As part of this, ciphers are checked, certificate validation is done.

- The SSL Server side of the communications MUST have a personal certificate. Information from this certificate is passed to the SSL client side, so it can be validated to identify the SSL server and validate that it is "trusted."

- The SSL client side may also have a certificate. This may or may not be presented back to the SSL server side. If it is, it is then validated on the SSL Server side to ensure it is valid and "trusted."

There is a configuration option in MQ, SSLCAUTH (SSL Client 'certificate needed for' Authentication); if this is set to REQUIRED, the SSL client MUST have and present its personal certificate to the SSL Server for validation. The default value for SSLCAUTH is REQUIRED.

SSLCAUTH(OPTIONAL) means that the client does not have to send it's certificate; however, if it does then this certificate must still be trusted by the Server. If you change SSLCAUTH to OPTIONAL and your connections work then you know it means that the client is not sending a certificate; while if it still does not work the issue could be elsewhere or that the certificate is not trusted by the server.

If you are having issues, you may want to set SSLCAUTH to OPTIONAL to see if the connection works, or if it still fails. This can help narrow down the issue. You can easily change it back to REQUIRED.

As part of the SSL initialization, on the machine requiring SSL (the SSL Server), the SSL Server's side of the communication MUST have a personal certificate.

The SSL server's personal certificate is validated on both sides of the connection, first locally on the SSL server side, then later information from it is passed to the remote side and some validation is done there as well.

IMPORTANT: Validating the certificate chain:

> On the SSL server side, this involves checking to see that a complete chain of certificates (from the personal certificate to a trusted CA root certificate) is in the keystore. When some signer certificates are missing this causes many issues/problems. Each certificate in the certificate chain is validated and must be present on the SSL Server side.
>
> On the remote side, when the remote certificate is validated, you do not always need the full chain of certificates to the Root CA signer. This is dependent on the SSL validation implementation.
>
> - If the SSL client is using GSKit, then only the Root CA signer certificate is needed.
> - If the SSL client is using some other SSL implementation, ie: Java/JSSE, or possibly .NET in managed mode, the full chain of certificates leading to and including the Root CA certificate is validated.
> - It is safer to exchange the full signer chain, but GSKit won't necessarily use all the certificates in the keystore for validation.

A simplistic view of a certificate:

Personal certificate contains:

- Subject/Issued to - Identification information, this is its Distinguished Name (DN = CN, ORG, C, etc),
- SubjectKeyIdentifier (hex code that identifies the cert)
- Issued by information, also made up of a DN
- AuthorityKeyIdentifier (hex code that identifies the signer/who this cert was Issued by)
- Private key
- Public key
- key signature algorithm (MD5, SHA-1, SHA-2,etc)
- other properties, AIA (AuthorityInfoAccess), valid dates, Usage information, etc..

Signer certificates contain similar:

- they do not contain the Private key
- if signed/issued by itself, it is a Root certificate. CA signer certs may be Root certificates. Also a self-signed certificate, which is actually a personal certificate signed by itself also fits this category.

Certificate requests, are unsigned personal certificates. When renewing a certificate, the certificate recreate request goes into the .rdb file (request database file)

Personal certificates contain the Private key. You never want to give out your private key.

Signer certificates, or public key certificates, are used to prove/validate ownership of a private key.

A Root certificate is a Signer certificate where the Subject/"Issued to" DN/SubjectKeyIdentifer) and "Issued by" DN/AuthorityKeyIdentifier are the same (ie: issued to itself). This includes a self-signed certificate. There is one additional thing a certificate needs to be a CA certificate (and this is the same for intermediate CA's). The certificate needs to have the BasicConstraints Usage Flag set so that it is a CA certificate; without it you will get validation errors.

Many folks are upgrading their certificates to SHA-2 signature algorithms and keysizes >=2048. This is primarily driven by the security industry, where it makes these more secure.

Basic MQ Management commands:

We've tried to standardize the commands in MQ to manage keystores and certificates.

I'm going to show commands from the command line. You can also use the IKeyMan GUI to manage your certificates.

runmqakm/runmqckm are the main commands we've standardized on in MQ v7.1+

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.sec.doc/q012670_.htm?lang=en

- runmqakm is c-based command line tool, both use mostly the same options/arguments. (similar to gsk7capicmd from old versions)
- runmqckm is java based command line tool (similar to gsk7cmd from old versions)

- runmqakm is faster, can create and handle certificates that use the Elliptic Curve algorithms but cannot create or access JKS key stores.
- runmqckm is slower, cannot handle Elliptic Curve certificates but can access JKS keystores.

Basic commands for managing keystore files:

https://developer.ibm.com/answers/questions/270930/for-mq-how-can-i-modify-a-jks-keystore-or-truststo/

Here are examples of 2 very common commands we need to manually check the certificates

To list the certificates in a keystore (certificate labels are listed):

```
runmqakm -cert -list -db <keystorefilename>
```

To show details of a specific certificate by the labelname:

```
runmqakm -cert -details -db  <keystorefilename> -label <labelname>
```

You can also provide a `'-pw <password>'` option to avoid being prompted for a password if one is needed.

Simplified MQ certificate process:

The overall process is similar if using self-signed or CA-signed certificates (internal or external CA)

CA -signed

1) Make cert request
2) Send cert req file to the CA to get signed.. (internal or external)
3) CA sends back
   a) signed cert req
   b) CA signers root and any intermeds
4) '-cert -add' CA signers
5) '-cert -receive' signed cert req
6) RUNMQSC: refresh security type (ssl)
7) Exchange the CA signers with any remote sides for validation

self-signed

1) Create certificate
2) it is already signed..
3) Extract the public signer, '-cert -extract'
4) - signer is already part of the self-signed certificate
5) self-signer is already signed
6) RUNMQSC: refresh security type (ssl)..
7) Exchange the extracted cert-public-signer with any remote sides for validation

In the MQ IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.sce.doc/q014180_.htm?lang=en

Troubleshooting Keystore/Certificate issues:

Items to check:

I - Certificate keystore exists and is valid/accessable
II - Certificates exist, certificate names
III - Certificate chain to a CA root certificate exists and is valid**
IV - Certificates are marked as "Trusted"
V - Certificate dates are good
VI - Certificates are not revoked
VII - Remote certificate passed during SSL negotiation also checked
VIII – Ensure the correct signer certificates were exchanged..

If any of these fail then SSL communication will fail and be closed. This usually results in a 2393 MQRC_SSL_INITIALIZATION_ERROR error.

Depending on which side has an issue, the error log may have some additional details.

Some of the checks are pretty straight forward. Most all of these checks are done by checking the certificate details.

List certificate labels in the keystore:
`runmqckm -cert -list -db (keystore-filename)` = lists all certificate labels in the keystore
`runmqckm -cert -list personal -db (keystore-filename)` = lists personal cert labels in the keystore
`runmqckm -cert -list CA -db (keystore-filename)` = lists signer certificate labels in the keystore

List certificate details:
```
runmqckm -cert -details -db (keystore-filename) -label (labelname)
```

(now go through each of the items above)

I - Certificate keystore exists and is valid/accessible

There are various ways to identify the location of these keystores: Environment variables, Java system properties, in code, etc..

The MQ product documentation contains much information on this:

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.sec.doc/q012660_.htm?lang=en

SSLKEYR - queue manager property

For a queue manager, in RUNMQSC:
```
DISPLAY QMGR SSLKEYR
```

For a client, can be set in an environment variable
MQSSLKEYR - client side environment variable

or can be set in code
export MQSSLKEYR=/home/mqapp/mqappkey

if files are in /home/mqapp, named.. mqappkey.kdb, mqappkey.sth, mqappkey.rdb..
(.kdb = key database file, .sth = password stash file, .rdb = cert request db file)

These point to the ***base file-name*** of the keystore and do NOT include the file extension (usually .kdb)

Also check file permissions. userid of the running app or qmgr needs to have read access.

For Java/JMS programs, the keystore can be set via JVM options and in code.
Java programs require a .jks (Java Keystore file). For Java, the SSL functionality is actually provided by the JVM/JRE.

II - Certificates exist, Certificate names are correct

> Personal certificates are searched for by the labelname

Before MQ v8, for the queue manager and MQ Client application (non-Java), the personal certificate had a required certificate label name:
for queue manager: ibmwebspheremq followed by lower-case queue manager name
for user MQ app: ibmwebspheremq followed by lower-case userid of user running the MQ app

** - A common issue is that the certificate label exists, but the certificate was added with '-cert -add' command. The '-cert -add' command adds the PUBLIC part of the certificate as a signer certificate. Certificates added in this way are NOT personal certificates. They will NOT have a private key. List the personal certificates to verify it is actually a personal certificate. '-cert -receive' matches the signed certificate request with a certificate request in a keystore to make a personal certificate. Also 'cert -export' and '-cert -import' can be used to move a personal certificate.

In MQv8, the certificate label can be set, based on the CERTLABL property, both on qmgr and on the channel.

Before MQ v8, the label name for a digital certificate to be used by an MQ Client was fixed by MQ. You had to label your certificate exactly as MQ required for the certificate to be found.

In MQv8 Client application, the client can name Client Certificate in several ways.

- mqclient.ini file SSL Stanza
  CertificateLabel
- MQCONNX (MQSCO structure)
  CertificateLabel
- Environment variable
  export MQCERTLABL=MyCert

> Signer certificates are used to validate personal certificates. They are searched for by the DN (distinguished name), ie: CN=Mike, O=IBM, C=US

When searching for signer certificates, if there are multiple signer certificates with the same DN, the first found is returned. If this is not the correct one it may fail.

III - Certificate chain to a CA Root certificate exists and is valid

Recall the ***IMPORTANT: Validating the certificate chain*** from pg 6.

On the SSL server side, the full chain of certificates from the personal certificate to a Root certificate is validated.

To see that there is a complete chain of certificates from the personal certificate to a trusted CA root certificate, SSL code searches for a chain of certificates, starting with the personal certificate and checking for the certificate who it was "Issued by", and so on, until it gets to a Root certificate.

Much of the MQ documentation provides examples split into 2 ways, 1) using self-signed certificates and 2) using CA-signed certificates.

A self-signed certificate is really just a simple case, where the certificate is both a personal certificate and its own CA/signer root certificate.

Examples:

QM1 qmgr cert chain (we will see this example in detail later)
- ibmwebspheremqqm1 personal certificate for qmqr
  issued/signed by IBM CA intemediate - ibmCAintermed
    issued/signed by IBM CAroot - ibmCAroot**

QM2 qmgr cert chain:
- ibmwebspheremqqm2 personal certificate for qmqr
  issued/signed by Entrust Intermed1 - EntrustCAintermed1
    issued/signed by Entrust Intermed2 - EntrustCAintermed2
      issued/signed by Entrust CA Root -EntrustCAroot**

QM3 qmgr cert chain: (self signed – special case)
- ibmwebspheremqqm3 personal certificate for qmqr
  issued/signed by itself (issuer and subject are both this qmgrs identity.)

The certificate chain is probably the hardest part to validate yourself. You need to check to ensure that the keystore with the personal certificate (queue manager's or user's) has its certificate and also all the signer certificates in its certificate chain which lead back and include the root signer certificate. Each one must be validated.

In MQ v8, mqcertck, can be used to check this automatically
- new command
- mqcertck.stdout in runmqras for MQ v8 (8.0.0.4+)
- mqcertck can be used to also check the client's keystore.

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.ref.adm.doc/q120895_.htm?lang=en

Case of self-signed is simple as the certificate is a personal certificate and also its own root signer certificate.

Case of a CA-signed certificate gets a little harder
You need to verify that you have all the 'correct' certificates in the certificate chain
Start with the personal certificate
- check these by
  - first searching for the issuer of the certificate. Search the subject of the signer certificates (by DNs, since this is how they are searched/retrieved)
  - then when found, to ensure it is the correct one by comparing the AuthorityKeyIdentifier of the certificate to the SubjectKeyIdentifier of the signer certificate
  - if they match, then continue.If they fail then SSL FAILS.
  - continue until you get to the CA root certificate.
    remember, CA root certificate Subject and Issuer are the same.

Also check that Trusted = enabled and the Dates are ok as you go through each.

IV - Certificates are marked as "Trusted"

(done while checking the certificate details/chain)

If a certificate is NOT trusted, it is not available for use. This can cause the certificate chain to be invalid. Even though it is in the keystore, it is not used; thus if it is needed in the certificate chain, it would be missing.

V - Certificate dates are good.

(done while checking the certificate details/chain)

This is pretty obvious, we need to make sure the certificate is valid based on the dates in the certificate.

VI - Certificates are not revoked.

Depending on the configuration, calls may be made automatically to check if the certificates are still valid, or if they are added to a revocation list.

These checks are done depending on some qm.ini settings (example: OCSPAuthentication) or due to AUTHINFO records of type CRLLDAP and OCSP. Settings in the certificates can also affect the OCSP/CRL checking.

If an issue occurs when checking if a certificate is revoked, there is usually an error in the queue manager error logs which would indicate if there was an issue checking the OCSP responders.

By default OCSP authentication (OCSPAuthentication) is REQUIRED. With this setting, if the checks result in an UNKNOWN status, the OCSP check will fail and the channel will terminate. If OCSPAuthentication is set to WARN we will output a warning in the error logs.If it is OPTIONAL then we just accept it.

If an error points in this direction, it is easy to temporarily disable and test if this is the cause of the issue. You can easily re-enable it later.

Many times servers are not allowed to get to the internet via http. The OCSP checking can be a factor of the actual CA certificates. They can define how/where this is done.

To disable the OCSP/CRL checking
Add an SSL stanza to your queue manager's qm.ini, or the client's mqclient.ini:

SSL:
  OCSPCheckExtensions=NO
  OCSPAuthentication=OPTIONAL
  CDPCheckExtensions=NO

for Queue manager:
http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.con.doc/q019040_.htm?lang=en

for MQ client:
http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.con.doc/q016900_.htm?lang=en

There are times when a OCSP HTTP proxy is needed to get to the OCSP responder servers. If your network team says this is available it can be configured via the following setting:

SSLHTTPProxyName=

Your queue manager or client does need to be stopped and restarted to have these changes take effect.

AUTHINFO objects can also be defined for a queue manager to contain definitions required to perform certificate revocation checking using OCSP or Certificate Revocation Lists (CRLs) on LDAP servers. You may also need to disable this.

http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.ref.adm.doc/q085490_.htm?lang=en

If all pass, the SSL communication continues.

VII - Remote certificate passed during SSL negotiation is validated.

Similar checks done for the remote Certificate, which is passed during SSL negotiation.

Again, recall the ***IMPORTANT: Validating the certificate chain*** on pg 6.

When the SSL client is using GSKit, this involves checking that the Root signer certificate of the remote certificate exists and is valid in the local keystore. (This was done when you initially setup SSL, exchanging the root signer or CA root signer certificates to the remote partner).

If using some other SSL implementation (Java/JSSE/.NET managed, etc) then the full chain of signer certificates may be required in the keystore and validated.

SUMMARY:

All these checks MUST succeed for SSL communications to work. If any of these fail, SSL communication will fail and the MQ channel will be terminated.

We have been trying to improve the error messages at both ends. Always check the MQ queue manager or MQ client error logs.

Example QM keystore/certificate review:

Listing of certs:

```
>runmqakm -cert -list -db key.kdb -pw passw0rd

5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Certificates found
* default, - personal, ! trusted
!       Thawte Server CA
!       Thawte Premium Server CA
!       Thawte Personal Basic CA
!       Thawte Personal Freemail CA
!       Thawte Personal Premium CA
!       VeriSign Class 3 Public Primary Certification Authority - G5
!       VeriSign Class 1 Public Primary Certification Authority - G3
!       VeriSign Class 2 Public Primary Certification Authority - G3
!       VeriSign Class 3 Public Primary Certification Authority - G3
!       VeriSign Class 4 Public Primary Certification Authority - G3
!       VeriSign Class 1 Public Primary Certification Authority - G2
!       VeriSign Class 2 Public Primary Certification Authority - G2
!       VeriSign Class 3 Public Primary Certification Authority - G2
!       VeriSign Class 4 Public Primary Certification Authority - G2
!       VeriSign Class 1 Public Primary Certification Authority
!       VeriSign Class 2 Public Primary Certification Authority
!       VeriSign Class 3 Public Primary Certification Authority
!       Entrust.net Secure Server Certification Authority
!       Entrust.net Certification Authority (2048)
!       Entrust.net Client Certification Authority
!       Entrust.net Global Client Certification Authority
!       Entrust.net Global Secure Server Certification Authority
!       ibmca root
!       ibmca intermediate
-.......ibmwebspheremqtestibmca2

Just list personal certificates to verify:

>runmqckm -cert -list personal -db key.kdb -pw passw0rd
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Certificates in database E:\ssl\testibmca\testibmca2\key.kdb:
   ibmwebspheremqtestibmca2
```

In this case, the personal certificate is signed by IBM CA (Intermediate and Root):

Other things to be careful of here:

- If the keystore is old, some of the certificates in there may have expired or been re-issued.
- Older versions of MQ pre-populated many CA signer certificates. Newer versions of MQ do not do this.
- It is a good idea to periodically check/clean-up your keystores.

Details of queue manager personal certificate:

```
>runmqakm -cert -details -db key.kdb -pw passw0rd -label ibmwebspheremqtestibmca2

5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Label : ibmwebspheremqtestibmca2
Key Size : 2048
Version : X509 V3
Serial : 797edafddadad
Issuer : CN=IBM INTERNAL INTERMEDIATE CA,O=International Business Machines Corporation,C=US
Subject : CN=testibmca2,O=ibm.com,L=Poughkeepsie\, NY,ST=Poughkeepsie\, NY,C=US
Not Before : February 26, 2016 12:00:00 AM EST
Not After : February 24, 2019 11:59:59 PM EST
Public Key
    00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
..
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    F6 66 54 88 96 50 E5 FE 35 0E 36 5D B9 0D 47 65
    27 25 82 49
Fingerprint : MD5 :
    05 A5 B5 77 8C 74 4D C6 01 20 8C 42 A6 BB 3A F3
Fingerprint : SHA256 :
    44 CE 5A 9F BC 95 88 29 F5 7A 34 ED 48 2A 95 B5
    63 92 7F 30 FB 2A 5A F1 4F 44 83 3B 22 66 0C CB
Extensions
    key usage: digitalSignature, keyEncipherment
        critical
    eku
        serverAuth
        clientAuth
    CRLDistributionPoints
      fullname:
      directoryname:CN=CRL63,CN=IBM INTERNAL INTERMEDIATE CA,O=International Business Machines
Corporation,C=US
      fullname:
      uniformResourceID:GSKASNObject: OBJECT(tag=22, class=0)
value: -----BEGIN HEX-----
16 39 68 74 74 70 3A 2F 2F 64 61 79 6D 76 73 31     .9http://daymvs1
2E 70 6F 6B 2E 69 62 6D 2E 63 6F 6D 3A 32 30 30     .pok.ibm.com:200
31 2F 50 4B 49 53 65 72 76 2F 63 61 63 65 72 74     1/PKIServ/cacert
73 2F 43 52 4C 36 33 2E 63 72 6C                    s/CRL63.crl
-----END HEX-----

    SubjectKeyIdentifier
      keyIdentifier:
    02 87 CC 68 A2 4D 32 04 A4 40 29 19 FA 9A 26 5A
    37 37 76 34
    AuthorityKeyIdentifier
      keyIdentifier:
    06 2F DA C7 9A 1F 3F 87 51 B9 4D D0 F6 DA AE 97
    46 4A 9D C8
      authorityIdentifier:
      authorityCertSerialNumber:
    06 2F DA C7 9A 1F 3F 87 51 B9 4D D0 F6 DA AE 97
    46 4A 9D C8
    certificatePolicies
       any
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
Value
    00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
..
Trust Status : Enabled
```

Details of IBM Intermediate certificate:

```
>runmqakm -cert -details -db key.kdb -pw passw0rd -label "ibmca intermediate"

5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Label : ibmca intermediate
Key Size : 2048
Version : X509 V3
Serial : 11
Issuer : CN=IBM Internal Root CA,O=International Business Machines Corporation,C=US
Subject : CN=IBM INTERNAL INTERMEDIATE CA,O=International Business Machines Corporation,C=US
Not Before : February 25, 2015 12:00:00 AM EST
Not After : December 30, 2023 11:59:59 PM EST
..
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    F0 46 B4 00 B8 52 24 6E A2 94 6B 17 CE 83 23 49
    54 9A 3A 49
Fingerprint : MD5 :
    44 D6 30 B1 19 66 3A BD 16 0C 4E 52 4F 61 AF 98
Fingerprint : SHA256 :
    F0 4E 33 1A B5 4C 8D F0 ED E7 AB 1B 67 0B ED BB
    D1 3C 6F 52 4B D9 B4 09 F6 CA 03 AC 35 81 04 C8
Extensions
    key usage: keyCertSign, cRLSign
        critical
    basicConstraints
        ca = true
        pathLen = 1655248122
        critical
    CRLDistributionPoints
      fullname:
      directoryname:CN=CRL0,CN=IBM Internal Root CA,O=International Business Machines Corporation,C=US
      fullname:
      uniformResourceID:GSKASNObject: OBJECT(tag=22, class=0)
value: -----BEGIN HEX-----
16 3C 68 74 74 70 3A 2F 2F 64 61 79 6D 76 73 31     .<http://daymvs1
2E 70 6F 6B 2E 69 62 6D 2E 63 6F 6D 3A 32 30 30     .pok.ibm.com:200
31 2F 50 4B 49 53 65 72 76 2F 63 61 63 65 72 74     1/PKIServ/cacert
73 52 6F 6F 74 2F 43 52 4C 30 2E 63 72 6C           sRoot/CRL0.crl
-----END HEX-----

    SubjectKeyIdentifier
      keyIdentifier:
    06 2F DA C7 9A 1F 3F 87 51 B9 4D D0 F6 DA AE 97
    46 4A 9D C8
    AuthorityKeyIdentifier
      keyIdentifier:
    F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
    0E 6F D8 E3
      authorityIdentifier:
      authorityCertSerialNumber:
    F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
    0E 6F D8 E3
..
Trust Status : Enabled
```

Details of IBM CA certificate:

```
>runmqakm -cert -details -db key.kdb -pw passw0rd -label "ibmca root"

5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Label : ibmca root
Key Size : 2048
Version : X509 V3
Serial : 14
Issuer : CN=IBM Internal Root CA,O=International Business Machines Corporation,C=US
Subject : CN=IBM Internal Root CA,O=International Business Machines Corporation,C=US
Not Before : February 24, 2016 12:00:00 AM EST
Not After : January 2, 2035 11:59:59 PM EST
..
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    66 7C 48 44 D0 B6 0B EF 1A F7 ED D5 2D C3 55 76
    B0 1A 02 73
Fingerprint : MD5 :
    79 6A ED C8 E4 06 4D 9C 09 54 1D 25 08 8A 97 15
Fingerprint : SHA256 :
    EC 8B BD C4 2A 9C FD AF 7D 02 94 11 50 16 C2 A8
    2B F7 3E 6B 4C 24 46 0E 75 EC A9 FA A6 A2 42 EB
Extensions
     (2.16.840.1.113730.1.13)
        Value
    16 30 47 65 6E 65 72 61 74 65 64 20 62 79 20 74
    68 65 20 53 65 63 75 72 69 74 79 20 53 65 72 76
    65 72 20 66 6F 72 20 7A 2F 4F 53 20 28 52 41 43
    46 29
    key usage: keyCertSign, cRLSign
        critical
    basicConstraints
        ca = true
        pathLen = -404768469
        critical
    SubjectKeyIdentifier
      keyIdentifier:
    F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
    0E 6F D8 E3
    AuthorityKeyIdentifier
      keyIdentifier:
    F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
    0E 6F D8 E3
      authorityIdentifier:
      authorityCertSerialNumber:
    F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
    0E 6F D8 E3
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
..
Trust Status : Enabled
```

A few other items to note:

- In the personal and intermediate certs, the CRL setting/URL provided and OCSP checking.
- BasicConstraints ca = true (sometimes needed to identify as CA cert)
- Signature Algorithm : SHA256WithRSASignature
  if this is MD5 - certs signed using MD5 alg are rejected if using TLS 1.2 cipher protocol
  http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.sec.doc/q010240_.htm?lang=en

Details of chain validation:

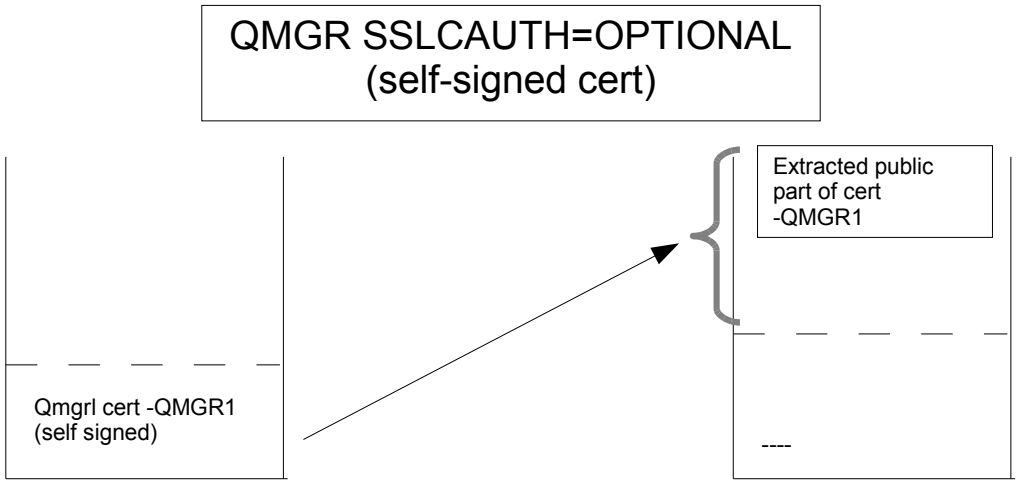
```
Extracting parts from the certificate details above
    –   first compare Issuer to Subject
    –   then verify the AuthorityKeyIdentifer to the SubjectKeyIdentifer

        First by DN (distinguished name)


        Label : ibmwebspheremqtestibmca2 (personal cert)
        Subject : CN=testibmca2,O=ibm.com,L=Poughkeepsie\, NY,ST=Poughkeepsie\, NY,C=US
        Issuer : CN=IBM INTERNAL INTERMEDIATE CA,O=International Business Machines
        Corporation,C=US


        Label : ibmca intermediate
        Subject : CN=IBM INTERNAL INTERMEDIATE CA,O=International Business Machines
        Corporation,C=US
        Issuer : CN=IBM Internal Root CA,O=International Business Machines Corporation,C=US


        Label : ibmca root (root cert)
        Subject : CN=IBM Internal Root CA,O=International Business Machines Corporation,C=US
        Issuer : CN=IBM Internal Root CA,O=International Business Machines Corporation,C=US


        Probably ok...


        verify by SubjectKeyIdentifier/AuthorityKeyIdentifier

        Label : ibmwebspheremqtestibmca2 (personal cert)
            AuthorityKeyIdentifier
              keyIdentifier:
            06 2F DA C7 9A 1F 3F 87 51 B9 4D D0 F6 DA AE 97
            46 4A 9D C8

        Label : ibmca intermediate
            SubjectKeyIdentifier
              keyIdentifier:
            06 2F DA C7 9A 1F 3F 87 51 B9 4D D0 F6 DA AE 97
            46 4A 9D C8
            AuthorityKeyIdentifier
              keyIdentifier:
            F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
            0E 6F D8 E3

        Label : ibmca root (root cert)
            SubjectKeyIdentifier
              keyIdentifier:
            F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
            0E 6F D8 E3
            AuthorityKeyIdentifier
              keyIdentifier:
            F9 DE 18 E5 9E 30 13 69 51 A7 FD 79 85 48 8C 7C
            0E 6F D8 E3

        In this case, Looks GOOD!!
```

Try mqcertck, as it does this automatically for you.

VIII – Ensure the correct signer certificates were exchanged

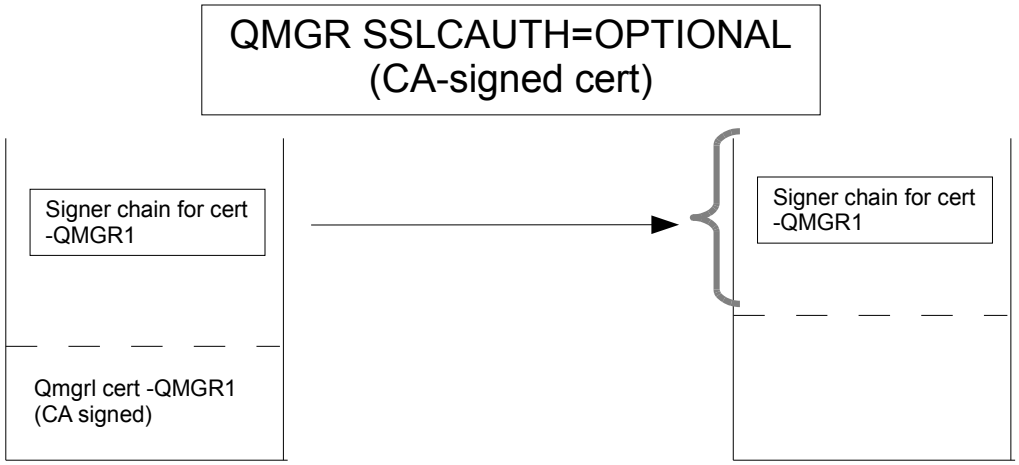
SSLCAUTH=OPTIONAL (client certificate optional)

IBM MQ - Certificates / Signers needed
In Keystores for SSL



QMGR1
(SSL server)
Label = ibmwebspheremq<qmgrname>
ie: ibmwebspheremqqmgr1
(all lowercase)

App or qmgr
(SSL client)

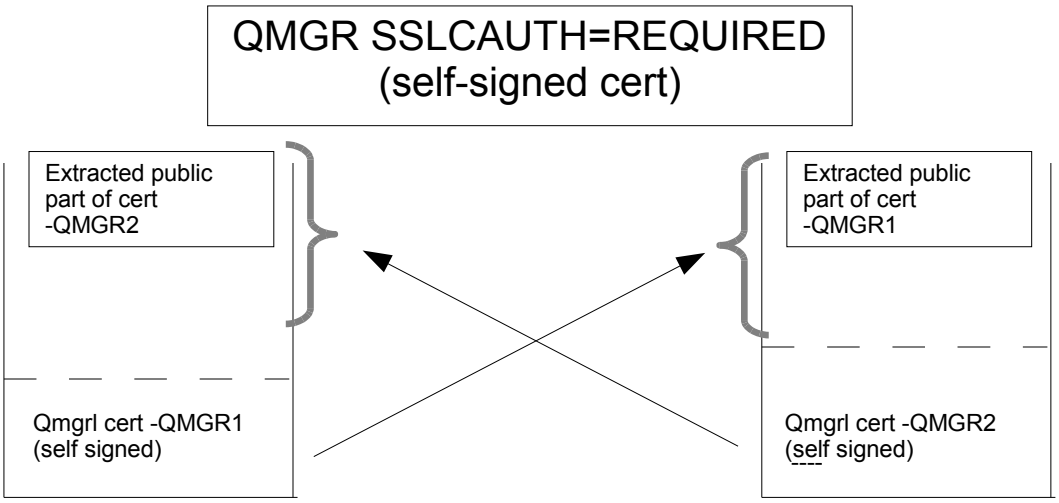


QMGR1
(SSL server)
Label = ibmwebspheremqqmgr1
Receive signed cert from CA,
add CA-signer certs.

App or qmgr
(SSL client)

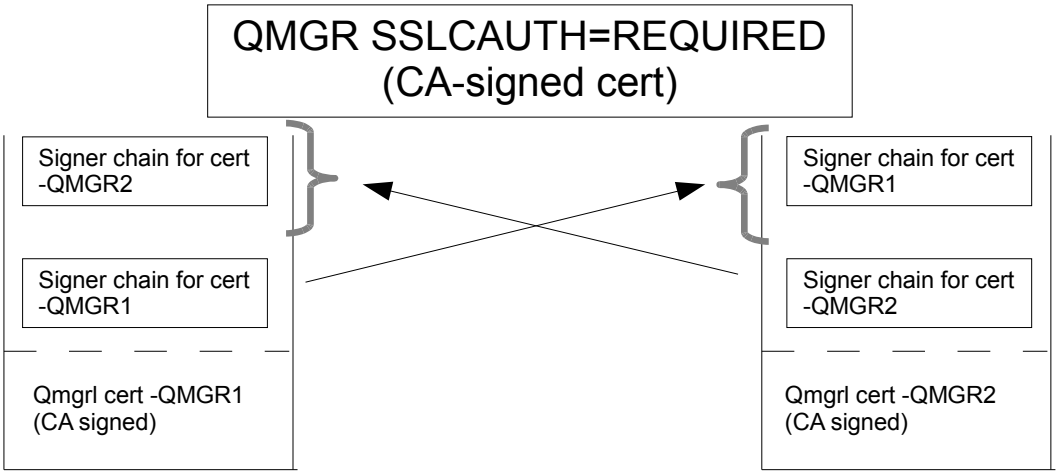
VIII – Ensuring the correct signer certificates were exchanged..
SSLCAUTH=REQUIRED (client certificate required)

IBM MQ - Certificates / Signers needed In Keystores for SSL



QMGR1
(SSL server)
Label = ibmwebspheremq<qmgrname>
ie: ibmwebspheremqqmgr1
(all lowercase)

App or qmgr
(SSL client)



QMGR1
(SSL server)
Label = ibmwebspheremqqmgr1
Receive signed cert from CA,
add CA-signer certs.

App or qmgr
(SSL client)

Diagnostics to collect if IBM support needed: (for SSL keystore/certificate issue)

For each side of the connection, Please collect the following:

- Runmqras data collection
  ```
  runmqras -qmlist <QMGRNAME> -section defs -pmrno <PMRNO>
  ```

  http://www.ibm.com/support/docview.wss?uid=swg21624944

  You can add '-ftp IBM' onto the runmqras to automatically ftp the diagnostics in to ECUREP, if your server has access to do so.

  The runmqras creates a zip file to send in with Error logs, and RUNMQSC definitions for review. Important things we will look at here, are the definitions of the QMGR, Channels, and the error logs.

- Please identify the channel that is being used.

- Detailed directory list of the keystore files
  example: (depends on location of keystore files)
  dir d:\ibm\mq\qmgrs\MYQMGR\ssl
  ls -l /var/mqm/qmgrs/MYQMGR/ssl

- If possible, send in keystore files for review; otherwise, we need certificate lists and details.

  List the certificates in a keystore (certificate labels are listed):
  ```
  runmqakm -cert -list -db <keystorefilename>
   (use runmqckm if .jks keystore)
  ```

  Show details of all certificate labels in the keystore, run this for each label:
  ```
  runmqakm -cert -details -db <keystorefilename> -label <labelname>
   (use runmqckm if .jks keystore)
  ```

Related Links and Information:

HOW-TOs:

There have been many documents which provide examples, including in MQ IBM Knowledge Center and on the web.

Here are some recently done step-by-step examples with commands.

How do I configure an MQ client c-based application like amqsputc/amqsgetc to connect to an MQ server with SSL?
https://developer.ibm.com/answers/questions/250184/how-do-i-configure-an-mq-client-c-based-applicatio/

How do I configure MQ Explorer to connect to an MQ server with SSL?
https://developer.ibm.com/answers/questions/214274/how-do-i-configure-mq-explorer-to-connect-to-an-mq/

How do I configure SSL between 2 MQ queue managers (Sender/Receiver channels)?
https://developer.ibm.com/answers/questions/250219/how-do-i-configure-ssl-between-2-mq-queue-managers/

How to renew a certificate:
https://developer.ibm.com/answers/questions/268240/how-to-renew-a-certificate-that-is-about-to-expire.html#answer-268241

Product documentation / Secure Scenarios:
http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.sce.doc/q005302_.htm?lang=en

SHA256 / TLS 1.2 issues:

MQ SHA-2 support::
http://www-01.ibm.com/support/docview.wss?uid=swg21639606

2393 - SHA256 ciphers
https://developer.ibm.com/answers/questions/212135/in-mq-why-is-ssl-not-working-when-trying-to-use-sh/

AMQ9771, 2393 SSL Initialization error from a MQ Java/JMS application when trying to use an TLS AES 256 cipher?
https://developer.ibm.com/answers/questions/189995/why-do-i-get-amq9771-2393-ssl-initialization-error/

In MQ, Can I use a TLS 1.2 / SHA256 / SHA-2 / AES_256 ciphersuite when connecting MQ Java / JMS or WAS application to a MQ queue manager?
https://developer.ibm.com/answers/questions/253821/in-mq-can-i-use-tls-12-sha256-sha-2-ciphers-when-c/

ERROR/RESOLUTION dwAnswers:

'AMQ9637: Channel is lacking a certificate'
https://developer.ibm.com/answers/questions/187722/on-mq-trying-to-enable-ssl-and-getting-error-amq96.html#answer-187723

'AMQ9633: SSL channels fail to establish a connection
https://developer.ibm.com/answers/questions/211744/why-is-mq-encountering-the-error-amq9633-after-set/

OCSP check issue, delays, AMQ9716: Remote SSL certificate revocation status check failed
https://developer.ibm.com/answers/questions/257804/getting-delay-or-amq9716-remote-ssl-certificate-re/

AMQ9640: SSL invalid peer name, channel
https://developer.ibm.com/answers/questions/192426/why-is-websphere-mq-connection-failing-when-enabli/

deprecated SSL 3.0 CipherSpecs
https://developer.ibm.com/answers/questions/207403/deprecated-ssl-30-cipherspecs/

AMQ9660: SSL key repository: password stash file absent or unusable
https://developer.ibm.com/answers/questions/206629/why-am-i-getting-amq9660-ssl-key-repository-passwo/

NoSuchAlgorithmException: SHA224withRSA Signature not available
https://developer.ibm.com/answers/questions/251057/how-to-resolve-issue-with-mq-v7x-java-client-getti/

In MQ, how do you rename the certificate label if the label name is incorrect
https://developer.ibm.com/answers/questions/187471/in-mq-how-do-you-rename-the-certificate-label-if-t/

How to renew a certificate that is about to expire in IBM MQ
https://developer.ibm.com/answers/questions/268240/how-to-renew-a-certificate-that-is-about-to-expire/

How can I re-enable SSLv3 or other deprecated weak ciphers in WebSphere MQ v8
https://developer.ibm.com/answers/questions/219682/how-can-i-re-enable-sslv3-or-other-deprecated-ciph/

What TLS cipherspecs/ciphersuites are supported when connecting from Oracle Java (non-IBM JRE) to MQ queue manager?
https://developer.ibm.com/answers/questions/178651/what-tls-ciphersuites-are-supported-when-connectin/

AMQ8251 : 32-bit GSKit component could not be loaded due to error 536908320; or AMQ6175, AMQ6256
https://developer.ibm.com/answers/questions/258568/in-mq-dspmqver-getting-error-amq8251-32-bit-gskit/