Using SSL TLS in MQ 9.3 to connect a JMS client to a queue manager in Linux, using self-signed certificates, 2-way authentication

https://www.ibm.com/support/pages/node/7142241

Date last updated: 15-Mar-2024

Angel Rivera, Rich Montoy
IBM MQ Support
https://www.ibm.com/products/mq/support
Find all the support you need for IBM MQ

+++ Objective +++

The objective of this document is to provide step-by-step details to:
- connect an MQ JMS client,
- to a single-instance queue manager running in Linux,
- using self-signed certificates (ok for Test queue managers, but not for Production, because no Certification Authority is involved, that is, no root certificates).
- 2-way authentication (client authenticates the queue manager, and the queue manager authenticates the client).
- using runmqckm (iKeycmd) from the command line, that is, not using the strmqikm (iKeyman) GUI.

For illustration purposes the following protocol will be used:
TLS 1.3 compliant: TLS_AES_128_GCM_SHA256

In this document, these terms are used interchangeably: SSL and TLS.
TLS is the successor to SSL, but the term "SSL" is used for historical reasons and the MQ tooling refers to SSL, even though it applies also to TLS.
The default behavior is for the MQ Client application to ask for the authentication of the MQ queue manager, this is called "1-way authentication".

The method of "2-way authentication" is when additionally, the MQ queue manager asks for the authentication of the MQ Client application. This is accomplished by specifying SSLCAUTH(REQUIRED) in the definition of the channel.

This tutorial shows all the steps for "2-way authentication" but identifies those steps that can be skipped if you are interested only in "1-way authentication".

The configuration for SSL requires many steps and at different locations.
One common source of confusion when doing the setup for SSL/TLS is:
In which side of the connection a certain step/command needs to be taken?
Is "Step A" done at the host of the MQ client application?
… Or at the host of the queue manager?

This tutorial tries to be very explicit in this respect, and hopefully the confusion could be avoided.

This tutorial provides an “extreme summary” with all the commands, without explanation. The idea is that you can copy these commands into a text editor, edit them to suit your needs and then copy/paste into the command prompts for Windows and Linux.

+ Why using "self-signed certificates"?

Because for novice users this is the easiest approach: it requires the least amount of effort and steps.

<u>In which conditions would be ok to use them?</u>
Self-signed TLS certificates are suitable for personal use or for applications that are used internally within an organization.
They are usually used for testing environments or low-risk internal networks only.

<u>In which conditions would NOT be ok to use them?</u>
Self-signed certificates are NOT suitable for software used by external users or high-risk or Production environments because the certificates cannot be verified with a Certification Authority (CA).

++ The sample SSLSampleJMS is used in this tutorial

The Java GitHub sample SSLSampleJMS.java is used for testing because it does not require a CCDT.
https://github.com/ibm-messaging/mq-tls-ssl-wizard/blob/master/com.ibm.mq.ssl-wizard/src/tlswizard/samples/SSLSampleJMS.java#L1
GitHub, SSLSampleJMS for IBM MQ

A zip / tar.gz file with the modified source code (the "package" statement was commented out) and the compiled class file of the modified code, is available in the web page show at the top of this cover page:

```
   SSLSampleJMS.java
   SSLSampleJMS.class
```

+ Linux:

Copy the tar.gz file into a directory such as:

```
  /home/mqm/ssl
```

Extract the files from the zip file:

```
  tar -zxvf SSLSampleJMS.tar.gz
```

+ Windows:

Copy the zip file into a directory such as:

```
  C:\WinTools
```

Extract/unzip the file SSLSampleJMS.zip

++ Requirements for the MQ JMS Client Application

The host / server / VM that has the MQ JMS Client Application needs to have:

a) A Java Runtime Environment (JRE).

a.1) If you install the complete MQ Client, then ensure to install the IBM JRE that is included.

Linux:

For example, if using Linux, then install the fileset:

```
MQSeriesJRE-9.3.0-2.x86_64
```

Windows:

The full path for "java.exe" is:

```
%MQ_JRE_PATH%\bin\java.exe
```

Where "MQ_JRE_PATH is setup by the MQ utility: setmqenv
For example:

```
C:\Program Files\IBM\MQ\java\jre\bin\java.exe
```

a.2) If you do not install the MQ Client, then you need to install a JRE.

For example, if using RHEL or compatible OS, as user root you could issue:

```
yum install java
```

As of Feb-2024, the following was installed.
Installing:
java-1.8.0-openjdk

b) The MQ classes for JMS jar files.

- If you install the complete MQ Client, then ensure to install the MQ Java fileset.
For example, if using Linux, then install the fileset:

```
MQSeriesJava-9.3.0-2.x86_64
```

- If you do not install the complete MQ Client, then you can download and install only the MQ classes for JMS.
For details see:

https://www.ibm.com/support/pages/node/6202719
Download, install and test IBM-MQ-Install-Java-All (com.ibm.mq.allclient.jar) and IBM-MQ-Java-InstallRA (wmq.jmsra.rar)

c) (Optional) Wrapper batch / script files to facilitate running the samples

See section:
++ Useful batch command and script files

d) If you want to compile, then you must install the MQ Software Development Kit (SDK).
For example, if using Linux, then install the fileset:

```
        MQSeriesSDK-9.3.0-2.x86_64
```

e) If you want to compile, then you must install the "javac" compiler

For example, if using RHEL or compatible OS, as user root you could issue:

```
        yum install java-1.8.0-openjdk-devel
```

As of Feb-2024, the following was installed.
Installing:
 java-1.8.0-openjdk-devel

++ Useful batch command and script files

+ Linux

Create/designate a directory in your PATH for having utilities, such as
  $HOME/bin

File: set-mq-inst1

This script will setup the environment variables for MQ for Installation1 and adds the directories for the MQ samples in the PATH and shows the version of java.

+ begin file - ignore this line

```
# Name: set-mq-inst1
# Purpose: to setup the environment to run MQ in Installation1
. /opt/mqm/bin/setmqenv -n Installation1
# Additional MQ directories for the PATH
export
PATH=$PATH:$MQ_INSTALLATION_PATH/java/bin:$MQ_INSTALLATION_PATH/samp/bin:$MQ_INSTAL
LATION_PATH/samp/jms/samples:
# Add local directory for running Java/JMS programs
export CLASSPATH=$CLASSPATH:.
# Display the full fix pack level of MQ
dspmqver -f 2
# Show full path of "java"
echo "Full path of java: " `which java`
# Show version of java
java -fullversion
# end
```

+ end file - ignore this line

File: runmqjms

This is a wrapper that will invoke "java" with the desired line arguments.
There are 4 options, and you need to ensure to have only 1 of them to be active and the other 3 to be commented out.
The 1st one does not involve tracing.

This is the contents of the utility:

+ begin file - ignore this line

```
# Name: runmqjms

# Wrapper to run MQ JMS Client applications
# Uncomment/commentout the desired option (number 1 does not use tracing)

echo "runmqjms: Begin"

## Option 1: (Simplest, Easiest) No tracing. Not using jms.config, nor properties
java -Djava.library.path=$MQ_JAVA_LIB_PATH $*

## Option 2: Using Java arguments for the command line (trace is shown in stdout)
# java -Djava.library.path=$MQ_JAVA_LIB_PATH -
Dcom.ibm.msg.client.commonservices.trace.status=ON $*

## Option 3: Using Java arguments for the command line (trace file in the MQ trace
directory, name: mqjms_PID.trc)
# java -Djava.library.path=$MQ_JAVA_LIB_PATH -
Dcom.ibm.msg.client.commonservices.trace.status=ON -
Dcom.ibm.msg.client.commonservices.trace.outputName=$MQ_JAVA_DATA_PATH/trace/mqjms_
%PID%.trc $*

## Option 4: Using jms.config file
# java -Djava.library.path=$MQ_JAVA_LIB_PATH -
Dcom.ibm.msg.client.config.location=file:/var/mqm/myjms.config $*

# end script
```

+ end file - ignore this line

Example run for the sample JmsProducer to put a message into queue Q1 in queue manager QM93LNX:

```
cd /opt/mqm/samp/jms/samples
runmqjms JmsProducer -m QM93LNX -d Q1
```

c.2) Windows

There are 2 batch command files that could be helpful to you:
 set-mq-inst1.bat
 runmqjms.bat

Create/designate a directory in your PATH for having utilities, such as
 C:\WinTools

<u>File: set-mq-inst1.bat</u>

This batch file will setup the environment variables for MQ for Installation1 and adds the directories for the MQ samples in the PATH and shows the version of java.
Note that if you installed MQ in a non-default directory, you will need to modify the statement below for "setmqenv" to use the appropriate full path and Installation name
For this script, the JRE that is shipped with MQ is being used.

+ begin file - ignore this line

```
REM Setup the environment to run MQ from Installation1
CALL "C:\Program Files\IBM\MQ\bin\setmqenv" -n Installation1

REM Adding Samples to the path

SET
PATH=%PATH%;%MQ_FILE_PATH%\Tools\c\Samples\Bin;%MQ_FILE_PATH%\Tools\c\Samples\Bin64
;%MQ_FILE_PATH%\Tools\jms\samples;%MQ_JAVA_INSTALL_PATH%\bin\

dspmqver -f 2

ECHO "Setting up IBM JRE from MQ"
SET PATH=%MQ_JRE_PATH%\bin;%PATH%;

SET CLASSPATH=%CLASSPATH%;.
java -fullversion
```

+ end file - ignore this line

File: runmqjms.bat

Create the following sample batch file which will invoke the runtime "java" with the proper library path.

Notice that there are 4 options (3 of them are for different ways to enable the MQ Java tracing).
Ensure to have only one commented out.
The default is Option 1, which is the simplest option, without doing any tracing.
If you want to do tracing, then you need to comment out Option 1, and uncomment one of the other options.

Example run for the sample JmsProducer to put a message into queue Q1 in queue manager QM93WIN:

```
cd C:\Program Files\IBM\MQ\tools\jms\samples
runmqjms JmsProducer -m QM93WIN -d Q1
```

This is the contents of the utility:

+ begin file - ignore this line

```
@echo off
REM Name: runmqjms.bat
REM Wrapper to run MQ JMS Client applications
REM Uncomment/commentout the desired option (number 1 does not use tracing)

rem 1: (Simplest, Easiest) No tracing. Not using jms.config, nor properties
java -Djava.library.path="%MQ_JAVA_LIB_PATH%"  %*

rem 2: Using Java arguments for the command line (trace file in local directory,
name: mqjms_%PID%.trc)
rem java -Djava.library.path="%MQ_JAVA_LIB_PATH%" -
Dcom.ibm.msg.client.commonservices.trace.status=ON %*

rem 3: Using Java arguments for the command line (trace file in the MQ trace
directory, name: mqjms_%PID%.trc)
rem java -Djava.library.path="%MQ_JAVA_LIB_PATH%" -
Dcom.ibm.msg.client.commonservices.trace.status=ON
Dcom.ibm.msg.client.commonservices.trace.outputName=%MQ_JAVA_DATA_PATH%\trace\mqjms
_%PID%.trc %*

rem 4: Using jms.config file
rem java -Djava.library.path="%MQ_JAVA_LIB_PATH%" -
Dcom.ibm.msg.client.config.location=file:/C:/w32tools/myjms.config  %*
```

+ end file - ignore this line

+ Auxiliary articles

a) The information in the "extreme summary" sections from this tutorial were duplicated into the following far shorter document:

https://www.ibm.com/support/pages/node/7118737
Summary of IBM MQ SSL TLS commands to connect clients and queue managers, using self-signed certificates, 2-way authentication

b) During the preparation of this article, some errors were encountered and eventually were addressed. The following document was created to capture the symptoms and the actions that were taken to fix the problems.

https://www.ibm.com/support/pages/6955529
Exploring some troubleshooting scenarios for SSL/TLS in IBM MQ

++ Configuration:

+ Windows (Client):
Hostname: finestra1
User: Administrator
MQ Client: 9.3.5 CD all components installed, including JRE and SDK

+ Linux (Client):
Hostname: c46968v1
User: mqm
MQ Client: using only 9.3.0.15-IBM-MQ-Install-Java-All.jar

+ Linux (Queue Manager):
Hostname: stmichel1
User: mqm
MQ Queue Manager: 9.3.0.15 LTS
Name: QMSTMTLS
Port: 1419
MQ administrative command to create it:

```
  crtmqm -u SYSTEM.DEAD.LETTER.QUEUE -p 1419 QMSTMTLS
```

- Hint when using testing systems (not recommended for Production)

For testing purposes, disable connauth and chlauth to make things easier regarding channel authentication records and requiring passwords during connect.

```
alter qmgr chlauth(disabled) connauth('')
refresh security type(connauth)
```

- crtmqm creates automatically a subdirectory called "ssl"

Linux:
```
  /var/mqm/qmgrs/QMGRNAME/ssl
```

Windows:
```
  C:\ProgramData\IBM\MQ\qmgrs\QMGRNAME\ssl
```

+ These are the steps for 2-way authentication and are explained later in this tutorial:

Step 1: Client: Create SSL client key database, type pkcs12 (jks)

Step 2: Client (Skip for 1-way): Create personal certificate

Step 3: Client (Skip for 1-way): Extract the public SSL client certificate

Step 4: Client (Skip for 1-way): Copy Windows certificate to the SSL server side in Linux
Copy/transfer the public/signer SSL certificate administrator.crt in ASCII mode from the Windows host to the Linux host.

Step 5: Server (Linux): Create SSL server key database (type CMS)

Step 6: Server (Linux): Create certificate

Step 7: Server (Linux): Extract the public SSL server certificate

Step 8: Server (Linux): Copy Linux certificate to the SSL client side in Windows
Copy/transfer the public/signer SSL certificate QMSTMTLS.crt in ASCII mode from the Linux host to the Windows host.

Step 9: Server (Linux) (Skip for 1-way): Add the Windows certificate to Linux key database

Step 10: Server (Linux): Run MQSC commands for SSL server side queue manager

Step 11: Client: Add the Linux certificate to the Windows key database

Step 12: Using sample SSLSampleJMS to test the sending of a message from Client (Linux) to Server (Linux)

+++ Extreme summary

+ Usage notes

It is recommended that you copy all the commands in this section and paste them in a plain text editor such as Notepad, which uses monospace font and no wrap around the lines.

Keep in mind that some commands are very long and even though are shown in several logical lines in this document, they need to be performed in one single long line.

Then make global replacements for the items such as key database, passwords, user name, queue manager, etc.

Finally, you can copy from Notepad each command and execute it in a command prompt for Window or for the remote host in Linux.

+ Extreme summary: Client in Windows

Step 1: Client (Windows): Create SSL client key database, type pkcs12 (jks)

Issue "setmqenv" to set the MQ environment variables, including the ones for running Java/JMS applications.

Create directory "ssl" (or "tls") such as:

```
 C:\ProgramData\IBM\MQ\ssl
```

Inside that directory create key database:

```
runmqckm -keydb -create  -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -pw clientpass -type
pkcs12 -stash
```

Step 2: Client (Windows) (Skip for 1-way): Create personal certificate

```
runmqckm -cert  -create  -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label
ibmwebspheremqadministrator  -dn "CN=administrator,O=IBM,C=USA" -size 2048
```

List the newly created SSL certificate in Windows

```
runmqckm -cert  -list    -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed
```

List the details of the personal certificate:

```
runmqckm -cert  -details -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label
ibmwebspheremqadministrator
```

+ Deleting a certificate:
If you need to delete the certificate, you can reuse the command that shows the “details” and replace “details” with “delete”:
For example, you can use as the base:

```
runmqckm -cert  -details -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed
```

```
-label ibmwebspheremqadministrator
```

Then replace “details” for “delete”:

```
runmqckm -cert  -delete -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed
-label ibmwebspheremqadministrator
```

Step 3: Client (Windows) (Skip for 1-way): Extract the public SSL client certificate

This step is only needed when doing “2-way authentication”.
But it is not needed when doing “1-way authentication” (but it does not hurt if you do it).

The "extract" action gets the public key (signer) of a certificate from the database, but does NOT extract the private key.

The following command will create a file in the current directory.
For this tutorial this is the current directory so far:
C:\ProgramData\IBM\MQ\ssl

```
runmqckm -cert  -extract -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label
ibmwebspheremqadministrator  -target administrator.crt -format ascii
```

Step 4: Client (Windows) (Skip for 1-way): Copy Windows certificate to the SSL server side in Linux

This step is needed when doing “2-way authentication”.
It is not needed when doing “1-way authentication” (but it does not hurt if you do it).

Exchange keys with the host that has the queue manager.

Perform steps the Server (Linux).

Step 11: Client (Windows): Add the Linux certificate to the Windows key database

```
C:\ProgramData\IBM\MQ\ssl>
runmqckm -cert  -add     -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label
ibmwebspheremqqmstmtls -file QMSTMTLS.crt  -format ascii
```

List the certificates

```
C:\ProgramData\IBM\MQ\ssl>
runmqckm -cert  -list    -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed
```

Step 12: Using sample SSLSampleJMS to test the sending of a message from Client (Windows) to Server (Linux)

Go to the directory where the sample is stored.
For this tutorial is:

```
   cd c:\wintools
```

```
+ 1-way authentication: use channel JMSSSL1.SVRCONN

C:\WinTools> java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 C:\ProgramData\IBM\MQ\ssl\key.jks
clientpass

+ 2-way authentication: use channel JMSSSL2.SVRCONN

C:\WinTools> java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL2.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 C:\ProgramData\IBM\MQ\ssl\key.jks
clientpass
```

<u>**+ Extreme summary: Queue manager in Linux**</u>

As user root: Facilitate the remote access from Windows user "Administrator", truncated to 12 characters in lowercase: "administrato"

```
  As user root: Add group for non-MQ administrators
  groupadd -g 1005 mqusers

  useradd -u 603 -g mqusers -s /bin/bash -d /home/administrato -m administrato
```

As user mqm:

```
. /opt/mqm/bin/setmqenv -n Installation1

setmqaut -m QMSTMTLS -t qmgr                               -g mqusers +connect +inq +dsp
setmqaut -m QMSTMTLS -t q -n SYSTEM.DEFAULT.MODEL.QUEUE    -g mqusers +inq +browse +get
+dsp
setmqaut -m QMSTMTLS -t q -n SYSTEM.ADMIN.COMMAND.QUEUE    -g mqusers +inq +put +dsp
setmqaut -m QMSTMTLS -t q -n SYSTEM.MQEXPLORER.REPLY.MODEL -g mqusers +inq +browse +get
+dsp +put
setmqaut -m QMSTMTLS -t q -n Q1                            -g mqusers +inq +browse +get
+dsp +put
```

Step 5: Server (Linux): Create SSL server key database

```
runmqckm -keydb -create  -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -pw serverpass -
type cms -stash
```

Step 6: Server (Linux): Create certificate

```
runmqckm -cert  -create  -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed -label
ibmwebspheremqqmstmtls -dn "CN=QMSTMTLS,O=IBM,C=USA" -size 2048

runmqckm -cert  -list    -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed

runmqckm -cert  -details -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed -label
ibmwebspheremqqmstmtls
```

Step 7: Server (Linux): Extract the public SSL server certificate

```
runmqckm -cert  -extract -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed -label
ibmwebspheremqqmstmtls -target QMSTMTLS.crt -format ascii
```

Step 8: Server (Linux): Copy Linux certificate to the SSL client side in Windows
Copy/transfer the public/signer SSL certificate QMSTMTLS.crt in ASCII mode from the Linux host to the Windows host.

Exchange keys with the Client (Windows).

Step 9: Server (Linux) (Skip for 1-way): Add the Windows certificate to Linux key database

```
runmqckm -cert  -add     -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed -label
ibmwebspheremqadministrator  -file   administrator.crt  -format ascii

runmqckm -cert  -list    -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed
```

Step 10: Server (Linux): Run MQSC commands for SSL server side queue manager

Note:
There are 2 channels for 2-way authentication used in this tutorial:
JMSSSL2.SVRCONN for the Windows client, userid "administrator"
JMSSSL2LNX.SVRCONN for the Linux client, userid "mqm"

```
runmqsc QMSTMTLS
ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS')

DEFINE CHANNEL('JMSSSL2.SVRCONN') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
SSLCIPH(TLS_AES_128_GCM_SHA256) SSLCAUTH(REQUIRED) +
SSLPEER('CN=administrator,O=IBM,C=USA')

DEFINE CHANNEL('JMSSSL2LNX.SVRCONN') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
SSLCIPH(TLS_AES_128_GCM_SHA256) SSLCAUTH(REQUIRED) +
SSLPEER('CN=mqm,O=IBM,C=USA')

REFRESH SECURITY TYPE(SSL)

DEFINE QLOCAL(Q1) REPLACE

END
```

+ For “1-way authentication”:

```
runmqsc QMSTMTLS
ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS')

DEFINE CHANNEL('JMSSSL1.SVRCONN') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
SSLCIPH(TLS_AES_128_GCM_SHA256) SSLCAUTH(OPTIONAL) +
SSLPEER('')

REFRESH SECURITY TYPE(SSL)

DEFINE QLOCAL(Q1) REPLACE

END
```

Proceed with rest of steps from Client (Windows)

+ Extreme summary: Commands for a Linux Client

For completeness, this section provides the commands for a Linux Client.

User "mqm" creates directory to keep SSL related files:

```
mkdir /var/mqm/ssl
```

Setup the environment variables for MQ

```
.  /opt/mqm/bin/setmqenv -n Installation1
```

Step 1: Client (Linux): Create SSL client key database, type pkcs12 (jks)

Issue "setmqenv" to set the MQ environment variables, including the ones for running Java/JMS applications.

```
runmqckm -keydb -create  -db /var/mqm/ssl/key.jks -pw clientpass -type pkcs12 -stash
```

Step 2: Client (Linux) (Skip for 1-way): Create personal certificate

```
runmqckm -cert  -create  -db /var/mqm/ssl/key.jks -stashed -label ibmwebspheremqmqm -dn
"CN=mqm,O=IBM,C=USA" -size 2048

runmqckm -cert  -list    -db /var/mqm/ssl/key.jks -stashed

runmqckm -cert  -details -db /var/mqm/ssl/key.jks -stashed -label ibmwebspheremqmqm
```

Step 3: Client (Linux) (Skip for 1-way): Extract the public SSL client certificate

This step is only needed when doing “2-way authentication”.

```
cd /var/mqm/ssl
runmqckm -cert  -extract -db /var/mqm/ssl/key.jks  -stashed -label ibmwebspheremqmqm  -
target mqm.crt -format ascii
```

Step 4: Client (Linux) (Skip for 1-way): Copy Client Linus certificate to the SSL server side in Linux

Copy/transfer the public/signer SSL certificate administrator.crt in ASCII mode from the Client Linux host to the Server Linux host.

Perform steps the Server (Linux).

Step 11: Client (Linux): Add the Server Linux certificate to the Client Linux key database

```
runmqckm -cert  -add     -db /var/mqm/ssl/key.jks -stashed -label ibmwebspheremqqmstmtls
-file /var/mqm/ssl/QMSTMTLS.crt  -format ascii

runmqckm -cert  -list    -db /var/mqm/ssl/key.jks.kdb -stashed
```

Step 12: Using sample SSLSampleJMS to test the sending of a message from Client (Linux) to Server (Linux)

```
cd /home/mqm/ssl
```

1-way authentication:

```
$ java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks clientpass
```

2-way authentication:

```
$ java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL2LNX.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks
```

++ This tutorial uses the same database for both TrustStore and KeyStore

The **Server-side certificates** are certificates that the server presents to the client, and which are stored into a ***Trust Store*** of the client. Those are certificates that the client ***trusts***.

The **Client-side certificates**, which consist of a private key for the client to use and a corresponding public certificate that the client presents to the server, are put into a ***Key Store*** of the client. These are certificates that the server requires the client to send before accepting the connection. With client-side certificates the corresponding private key is also necessary so the client can actually encrypt data so it matches the certificate's public key.

The Trust Store and Key Store files have the same format, but they have different purposes.
You can put as many certificates as you like into a Trust Store and it will work; the client will "trust" all of them.
Putting multiple key/certificate pairs into a Key Store, however, is much less likely to work if you can't explicitly tell the client which key/cert to use with which connection, which you can't with the old IBM MQ steps.

The runmqckm command supports jks key stores.

You can use a single trust store for client-side certs and server-side certs if you want,
or you can separate them in a trust store and a key store.

++ Clarification of "extract"/"add" versus "export"/"import"

SSL uses public/private keys to provide a flexible encryption scheme that can be set-up at the time of the secure transaction.
When a certificate is created, it contains both the public and private keys.

The "extract" and "add" functions deal with ONLY the public keys.
That is, the "extract" gets the public key of a certificate from a database and the "add" puts the public key into a database.
The "extract" does NOT get the private key.
No passwords are required because the private key is not obtained.

The "export" and "import" functions deal with BOTH the public and private keys for a certificate.
Passwords are required due to the private key.

+++ Summary of steps: Client in Windows connecting to a queue manager in Linux

++ Step 1: Client (Windows): Create SSL client key database, type pkcs12 (jks)

+ Issue "setmqenv" to set the MQ environment variables, including the ones for running Java/JMS applications.

Some ways to issue setmqinst:

- If you have created the "set-mq-inst1.bat" batch file mentioned earlier in this document, you can issue:
C:\> **set-mq-inst1**

- Issue the command to specify the MQ environment variables:
C:\> **setmqenv -n Installation1**

- If necessary, you may need to specify the full path:
C:\> **"C:\Program Files\IBM\MQ\bin\setmqenv" -n Installation1**

Notice all the MQ environment variables set by setmqenv:

```
C:\> set MQ
MQ_DATA_PATH=C:\ProgramData\IBM\MQ
MQ_FILE_PATH=C:\Program Files\IBM\MQ
MQ_INSTALLATION_NAME=Installation1
MQ_INSTALLATION_PATH=C:\Program Files\IBM\MQ
MQ_JAVA_DATA_PATH=C:\ProgramData\IBM\MQ
MQ_JAVA_INSTALL_PATH=C:\Program Files\IBM\MQ\java
MQ_JAVA_LIB_PATH=C:\Program Files\IBM\MQ\java\lib64
MQ_JRE_PATH=C:\Program Files\IBM\MQ\java\jre
```

+ Create directory "ssl" (or "tls")

Go to the directory for the MQ Data, such as: cd C:\ProgramData\IBM\MQ\
cd %MQ_DATA_PATH%

Then create subdirectory for “ssl”:
C:\ProgramData\IBM\MQ> **mkdir ssl**
C:\ProgramData\IBM\MQ> **cd ssl**
You will be at the following directory:
C:\ProgramData\IBM\MQ\ssl

+ The following command creates a "jks keystore".

Even though the instances of some commands are shown in 2 lines, it is only a single line. You must ensure that if you copy/paste the command, you use one single long line:

```
runmqckm -keydb -create  -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -pw clientpass -type pkcs12
-stash
```

Notice that new files are created:

```
03/11/2024  12:33 PM             1,190 key.jks
03/11/2024  12:33 PM               193 key.sth
```

Note:
After stashing the password, for future SSL commands, it is better to use the -stashed command line option than to specify the -pw option.

++ Step 2: Client (Windows) (Skip for 1-way): Create personal certificate

This step is only needed when doing “2-way authentication”.

The step is not needed when doing “1-way authentication” (but it does not hurt if you do it).

+ Create the digital personal certificate.

Notice that the label and the CN include the user name in lowercase (administrator).
The value for the attribute "db" is the FULL name of the keystore file “key.jks”, including the suffix.

In recent versions of GSKit, the default signature algorithm is SHA256WithRSA and if this is ok with you, then you do not need to specify the parameter "-sig_alg SHA256WithRSA"

```
runmqckm -cert  -create  -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label
ibmwebspheremqadministrator  -dn "CN=administrator,O=IBM,C=USA" -size 2048
```

Where:
-label is the label name:
      ibmwebspheremqadministrator
   It is required to be the concatenation of:
      ibmwebspheremq + userid in lower case
   In this case:
      ibmwebspheremq + administrator
-dn is the "Distinguished Name", notice that for consistency, it is also in lowercase.
-size The recommended size is 2048 bits. The certificates with a size of 1024 are no longer recommended.

The key.jks is updated (notice that the size is bigger, because it has now 1 certificate):

```
03/11/2024  12:37 PM           3,771 key.jks
03/11/2024  12:36 PM             193 key.sth
```

+ List the newly created SSL certificate in Windows

runmqckm -cert -list   -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed

```
Certificates in database C:\ProgramData\IBM\MQ\ssl\key.jks:
   ibmwebspheremqadministrator
   dummy
```

You could specify an optional parameter for "-list" to show either "personal" or "ca" (Certification Authority), such as:

```
runmqckm -cert  -list personal  -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed
5724-H72 (C) Copyright IBM Corp. 1994, 2024.
Certificates in database C:\ProgramData\IBM\MQ\ssl\key.jks:
   ibmwebspheremqadministrator
```

```
runmqckm -cert  -list ca  -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed
5724-H72 (C) Copyright IBM Corp. 1994, 2024.
Certificates in database C:\ProgramData\IBM\MQ\ssl\key.jks:
   dummy
```

+ List the details of the personal certificate:

runmqckm -cert -details -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label ibmwebspheremqadministrator

<begin certificate – ignore this line>

```
Label: ibmwebspheremqadministrator
Key Size: 2048
Version: X509 V3
Serial Number: 65 EF 5D 63
Issued by: CN=administrator, O=IBM, C=USA
Subject: CN=administrator, O=IBM, C=USA
Valid: From: Monday, March 11, 2024 12:37:07 PM PDT To: Tuesday, March 11, 2025 12:37:07 PM PDT
Fingerprint:
    SHA1: 58:76:7F:9A:4C:46:D6:5F:EC:40:2F:D9:E6:8A:F8:EF:E0:B7:74:D2
    SHA256:
4E:E6:A7:7D:5C:6B:C3:CA:DC:25:F9:4B:A7:E7:65:85:60:2A:1F:BB:E6:4E:64:47:8B:BA:68:26:D1:10:06:07
    HPKP: iuwmzaJASWCjtFTXpxKyk5wjOsCNymuqNC1vnPtacHA=

Extensions:
  - AuthorityKeyIdentifier: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 59 d8 9f eb d2 00 65 3a  4c 9e fc b5 1c 2c c5 1d  Y.....e.L.......
0010: b7 c2 b4 ee                                        ....
]
]

  - SubjectKeyIdentifier: ObjectId: 2.5.29.14 Criticality=false
```

```
SubjectKeyIdentifier [
KeyIdentifier [
0000: 59 d8 9f eb d2 00 65 3a  4c 9e fc b5 1c 2c c5 1d  Y.....e.L.......
0010: b7 c2 b4 ee                                        ....
]
]

Signature Algorithm: SHA256withRSA (1.2.840.113549.1.1.11)
Trust Status: enabled
```

<end of certificate - ignore this line>

+ REFERENCE for deleting a certificate:

The following command is just for completeness, you do NOT need to issue it for this tutorial.

If you need to delete the certificate, you can reuse the command that shows the “details” and replace “details” with “delete”:

For example, you can use as the base:
runmqckm -cert -details -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label ibmwebspheremqadministrator

Then replace “details” for “delete”:
runmqckm -cert -delete -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label ibmwebspheremqadministrator

++ Step 3: Client (Windows) (Skip for 1-way): Extract the public SSL client certificate

This step is only needed when doing “2-way authentication”.

But it is not needed when doing “1-way authentication” (but it does not hurt if you do it).

The "extract" action gets the public key (signer) of a certificate from the database, but does NOT extract the private key.

The following command will create a file in the current directory.
For this tutorial this is the current directory so far:
C:\ProgramData\IBM\MQ\ssl

```
runmqckm -cert  -extract -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label
ibmwebspheremqadministrator  -target administrator.crt -format ascii
```

Notice that in the current directory a new file will be created, which is the extracted certificate which has the public key (but does NOT have the private key):

```
03/11/2024  12:42 PM            1,170 administrator.crt
03/11/2024  12:17 PM            3,771 key.jks
03/11/2024  12:08 PM              193 key.sth
```

This certificate file looks like this (only showing some lines though):

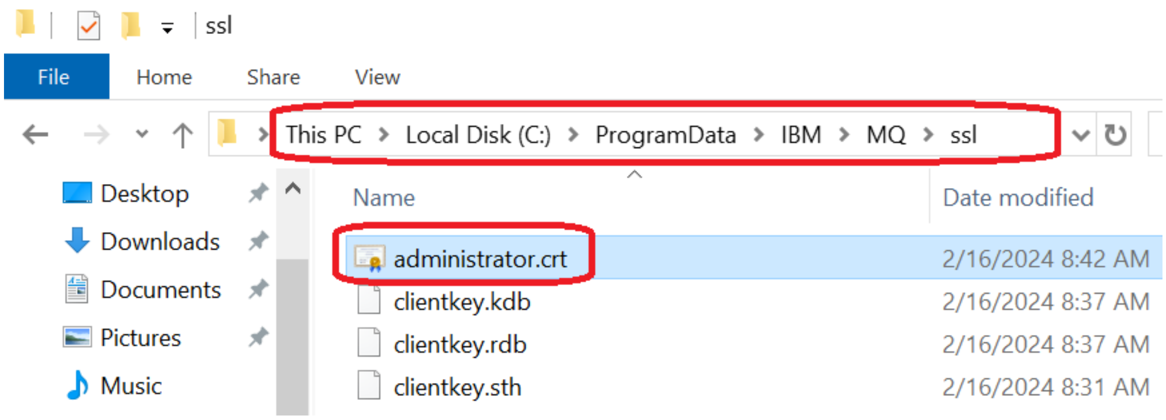
```
C:\ProgramData\IBM\MQ\ssl> type administrator.crt
-----BEGIN CERTIFICATE-----
MIIDEDCCAfigAwIBAgIEYi9uUTANBgkqhkiG9w0BAQUFADA0MQwwCgYDVQQGEwNVU0ExDDAKBgNV
BAoTA0lCTTEWMBQGA1UEAxMNYWRtaW5pc3RyYXRvcjAeFw0yMjAzMTQxNjMzMjFaFw0yMzAzMTQx
NjMzMjFaMDQxDDAKBgNVBAYTA1VTQTEMMAoGA1UEChMDSUJNMRYwFAYDVQQDEw1hZG1pbmlzdHJh
…
/SpdKi/wn0D8n5EEQ6Did+tiCvq6L9kzZVwYu5Xhy9HRDsusHbnlNCcP1Ysrjly5ESpRrnWucxcg
u7IavygoKS61806Y3ZZiYjEuFVNMIjuJccZrgDOQ6xwCm3rXFgiAfFQc/zdEtgQ=
-----END CERTIFICATE-----
```

+ Using Windows Explorer to browse the extracted certificate

You can open the Windows Explorer and display the files from the "ssl" directory where the certificate *.crt file was extracted.

Notice that the suffix ".crt" is recognized.

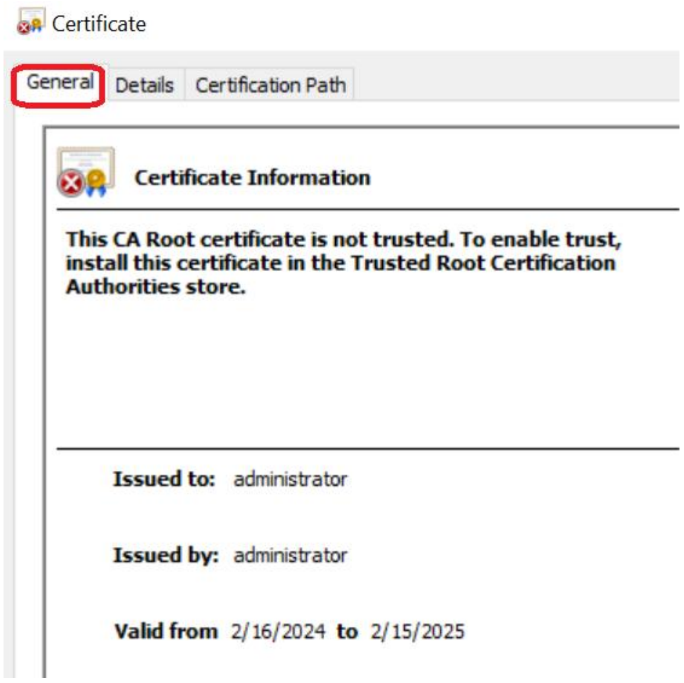
Double click on the file:
   administrator.crt



Name	Date modified
administrator.crt	2/16/2024 8:42 AM
clientkey.kdb	2/16/2024 8:37 AM
clientkey.rdb	2/16/2024 8:37 AM
clientkey.sth	2/16/2024 8:31 AM

You will see a dialog box with 3 tabs:

The tab "General" looks like this:



Certificate Information

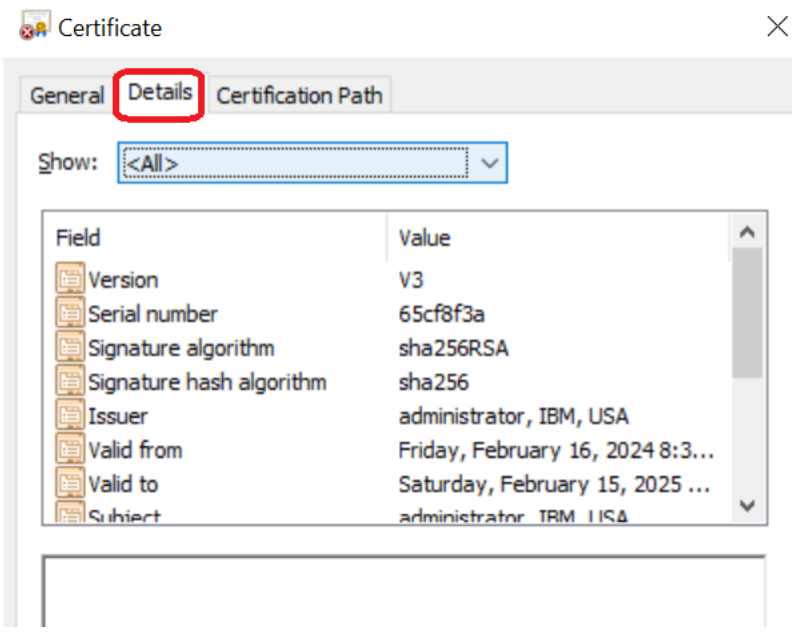
This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

Issued to: administrator

Issued by: administrator

Valid from 2/16/2024 to 2/15/2025

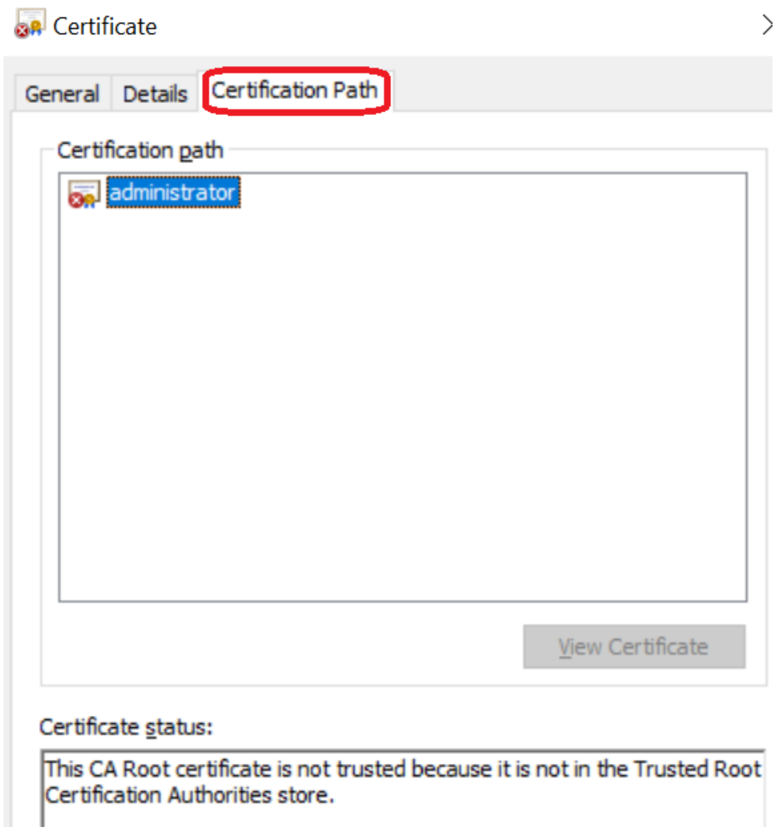
The tab "Details" looks like this:



Show: <All>

Field	Value
Version	V3
Serial number	65cf8f3a
Signature algorithm	sha256RSA
Signature hash algorithm	sha256
Issuer	administrator, IBM, USA
Valid from	Friday, February 16, 2024 8:3...
Valid to	Saturday, February 15, 2025 ...
Subject	administrator, IBM, USA

The tab "Certification Path" looks like this:



Certificate status:

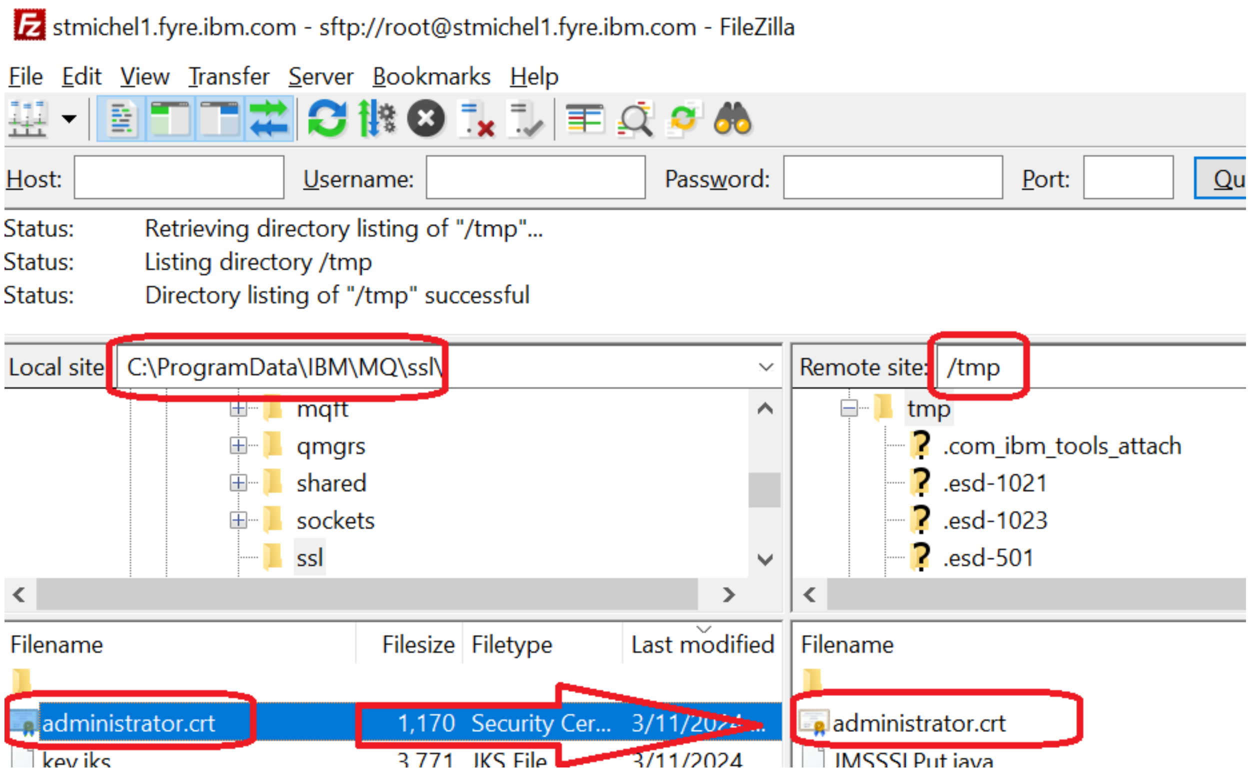
This CA Root certificate is not trusted because it is not in the Trusted Root Certification Authorities store.

++ Step 4: Client (Windows) (Skip for 1-way): Copy Windows certificate to the SSL server side in Linux

This step is needed when doing “2-way authentication”.
It is not needed when doing “1-way authentication” (but it does not hurt if you do it).

Copy/transfer the public/signer SSL certificate file administrator.crt in **ASCII mode** from the Windows host to the Linux host.

For this tutorial the utility “Filezilla” was used to copy the crt file into the Linux host at:
/tmp/administrator.crt


stmichel1.fyre.ibm.com - sftp://root@stmichel1.fyre.ibm.com - FileZilla

File Edit View Transfer Server Bookmarks Help
Host: Username: Password: Port: Qu

```
Status:    Retrieving directory listing of "/tmp"...
Status:    Listing directory /tmp
Status:    Directory listing of "/tmp" successful
```

Local site: C:\ProgramData\IBM\MQ\ssl\
Remote site: /tmp

mqft
qmgrs
shared
sockets
ssl
tmp
.com_ibm_tools_attach
.esd-1021
.esd-1023
.esd-501

Filename	Filesize	Filetype	Last modified
administrator.crt	1,170	Security Cer...	3/11/2024 ...
key.jks	3,771	JKS File	3/11/2024

Filename
administrator.crt
JMSSSLPut.java

++ Step 5: Server (Linux): Create SSL server key database (type CMS)

As user mqm, setup the environment variables for MQ:

```
  . /opt/mqm/bin/setmqenv -n Installation1
```

It is assumed that the queue manager QMSTMTLS has already being created and is using port 1419.

Notice that the subdirectory “ssl” is created by “crtmqm”.

```
mqm@stmichel1.fyre.ibm.com: /home/mqm
cd /var/mqm/qmgrs/QMSTMTLS/ssl
```

```
runmqckm -keydb -create  -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -pw serverpass -type cms -stash
```

Notice that 3 new files are created:

```
ls -l
-rw------- 1 mqm mqm  88 Mar 14 09:49 QMSTMTLS.kdb
-rw------- 1 mqm mqm  80 Mar 14 09:49 QMSTMTLS.rdb
-rw------- 1 mqm mqm 193 Mar 14 09:49 QMSTMTLS.sth
```

++ Step 6: Server (Linux): Create certificate

+ Create certificate

Starting with MQ 8.0, the queue manager's certificate does not need to be as in MQ 7.x:
 ibmwebspheremq + qmgrname
But we still recommend using that convention (Note: qmgrname should be in lowercase)

If the queue manager's label name is set to something else, the CERTLABL property of the queue manager must be set to the correct certificate labelname.
For more details see:
https://www.ibm.com/docs/en/ibm-mq/9.3?topic=attributes-channel-mqsc-keywords-c
```
IBM MQ / 9.3
Channel attributes for MQSC keywords (C)
CERTLABL (Certificate label)
```

If you follow the above convention, then you MUST USE the queue manager name in LOWERCASE for the label, such as:
 ibmwebspheremqqmstmtls

But for the CN, use the upper case name, such as:
 CN=QMSTMTLS

```
runmqckm -cert  -create  -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed
-label ibmwebspheremqqmstmtls -dn "CN=QMSTMTLS,O=IBM,C=USA" -size 2048
```

Notice that the key database is bigger, because it contains now 1 certificate:

```
-rw------- 1 mqm mqm 5088 Mar 14 09:50 QMSTMTLS.kdb
-rw------- 1 mqm mqm   80 Mar 14 09:50 QMSTMTLS.rdb
-rw------- 1 mqm mqm  193 Mar 14 09:49 QMSTMTLS.sth
```

+ List newly created SSL certificate in Linux

```
runmqckm -cert  -list    -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed
```

```
Certificates in database /var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb:
   ibmwebspheremqqmstmtls
```

+ List the details of the certificate.

runmqckm -cert -details -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed -label ibmwebspheremqqmstmtls

Excerpt:

```
Label: ibmwebspheremqqmstmtls
Key Size: 2048
Version: X509 V3
Serial Number: 62 2F 72 60
Issued by: CN=QMSTMTLS, O=IBM, C=USA
Subject: CN=QMSTMTLS, O=IBM, C=USA
Valid: From: Monday, March 14, 2022 9:50:40 AM PDT To: Tuesday, March 14, 2023 9:50:40 AM PDT
…
Trust Status: enabled
```

<u>++ Step 7: Server (Linux): Extract the public SSL server certificate</u>

The "extract" action gets the public key of a certificate from the database, but does NOT extract the private key.

The following command will create a file in the current directory.
In this scenario is:
   /var/mqm/qmgrs/QMSTMTLS/ssl

```
runmqckm -cert  -extract -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed
-label ibmwebspheremqqmstmtls -target QMSTMTLS.crt -format ascii
```

Notice that there is a new file, which has the certificate for the queue manager.

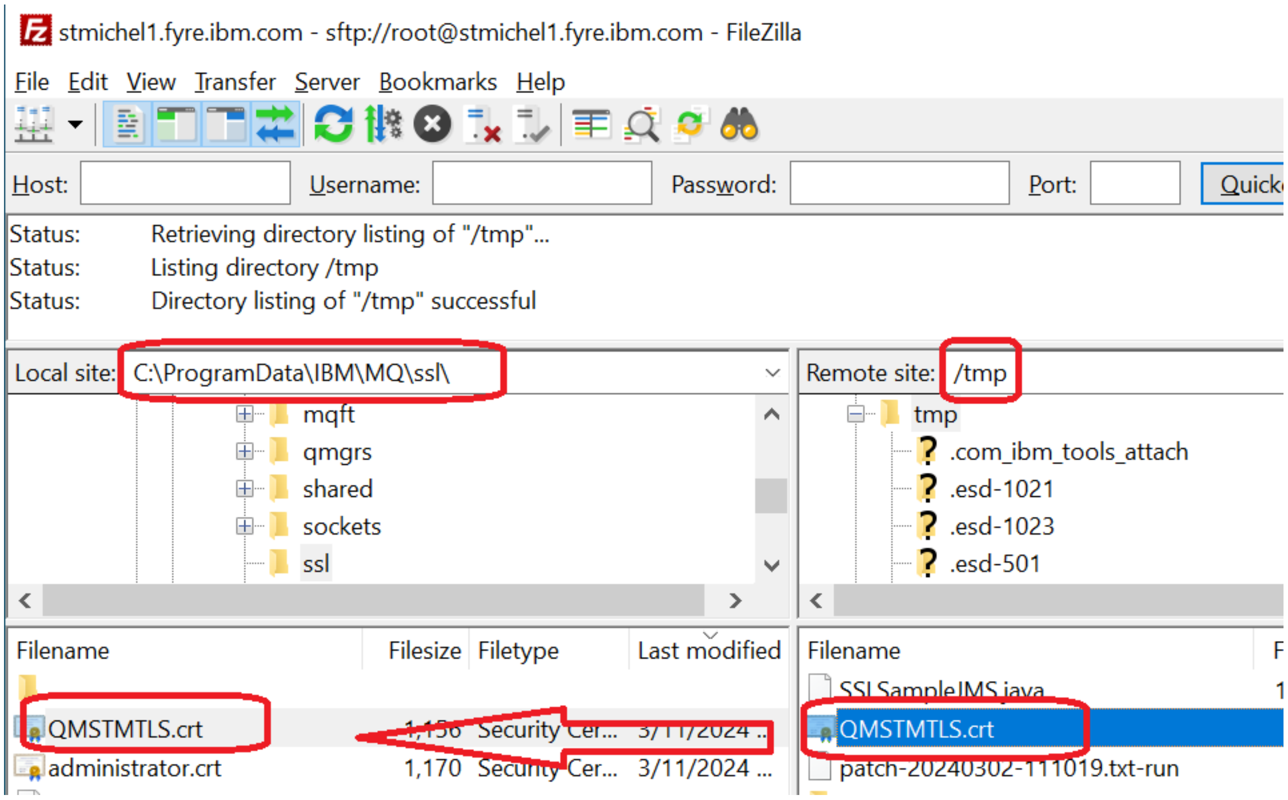
```
-rw------- 1 mqm mqm 1118 Mar 14 09:53 QMSTMTLS.crt
-rw------- 1 mqm mqm 5088 Mar 14 09:50 QMSTMTLS.kdb
-rw------- 1 mqm mqm   80 Mar 14 09:50 QMSTMTLS.rdb
-rw------- 1 mqm mqm  193 Mar 14 09:49 QMSTMTLS.sth
```

++ Step 8: Server (Linux): Copy Linux certificate to the SSL client side in Windows

Copy/transfer the public/signer SSL certificate QMSTMTLS.crt in ASCII mode from the Linux host to the Windows host.

Copy the QMSTMTLS.crt file from the "ssl" subdirectory into "tmp":
mqm@stmichel1.fyre.ibm.com: /var/mqm/qmgrs/QMSTMTLS/ssl
$ **cp QMSTMTLS.crt /tmp/.**

For this tutorial, we have already Filezilla running in the Windows host, thus, we will use that to copy the QMSTMTLS.crt from Linux into Windows.



stmichel1.fyre.ibm.com - sftp://root@stmichel1.fyre.ibm.com - FileZilla

```
Status:     Retrieving directory listing of "/tmp"...
Status:     Listing directory /tmp
Status:     Directory listing of "/tmp" successful
```

Local site: C:\ProgramData\IBM\MQ\ssl\

Remote site: /tmp

Filename	Filesize	Filetype	Last modified
QMSTMTLS.crt	1,156	Security Cer...	3/11/2024 ...
administrator.crt	1,170	Security Cer...	3/11/2024 ...

Filename
SSLSampleJMS.java
QMSTMTLS.crt
patch-20240302-111019.txt-run

File Edit View Transfer Server Bookmarks Help
Host: Username: Password: Port: Quick
mqft
qmgrs
shared
sockets
ssl
tmp
.com_ibm_tools_attach
.esd-1021
.esd-1023
.esd-501

+ For “2-way authentication”, in Linux, copy the administrator.crt file from /tmp into the ssl directory:

```
mqm@stmichel1.fyre.ibm.com: /var/mqm/qmgrs/QMSTMTLS/ssl
$ cp /tmp/administrator.crt .

$ ls -l
-rw-r--r-- 1 mqm mqm 1136 Mar 11 09:55 administrator.crt
-rw------- 1 mqm mqm 1118 Mar 11 09:53 QMSTMTLS.crt
-rw------- 1 mqm mqm 5088 Mar 11 09:50 QMSTMTLS.kdb
-rw------- 1 mqm mqm   80 Mar 11 09:50 QMSTMTLS.rdb
-rw------- 1 mqm mqm  193 Mar 11 09:49 QMSTMTLS.sth
```

+ In the Windows host, we have now:

```
C:\ProgramData\IBM\MQ\ssl>dir
03/11/2024  12:42 PM             1,170 administrator.crt
03/11/2024  12:17 PM             3,771 key.jks
03/11/2024  12:08 PM               193 key.sth
03/11/2024  12:56 PM             1,156 QMSTMTLS.crt
```

++ Step 9: Server (Linux) (Skip for 1-way): Add the Windows certificate to Linux key database

This step 2 is needed when doing “2-way authentication”.
That is, it is NOT needed when doing “1-way authentication”.

+ Add the public/signer certificate

```
runmqckm -cert  -add     -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed -label ibmwebspheremqadministrator  -file   administrator.crt  -format ascii
```

Notice that the size of the kdb became larger (it has now 2 certificates)

```
-rw-r--r-- 1 mqm mqm  1136 Mar 14 09:55 administrator.crt
-rw------- 1 mqm mqm  1118 Mar 14 09:53 QMSTMTLS.crt
-rw------- 1 mqm mqm 10088 Mar 14 09:58 QMSTMTLS.kdb
-rw------- 1 mqm mqm    80 Mar 14 09:58 QMSTMTLS.rdb
-rw------- 1 mqm mqm   193 Mar 14 09:49 QMSTMTLS.sth
```

+ List the certificates.

```
runmqckm -cert  -list    -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed
```

```
Certificates in database /var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb:
   ibmwebspheremqqmstmstls
   ibmwebspheremqadministrator
```

++ Step 10: Server (Linux): Run MQSC commands for SSL server side queue manager

+ NOTES regarding SSLPEER when using 2-way authentication:

Even though the SSLPEER attribute for the queue manager is optional, it is a good practice to use it for extra security.
Notice that the SSLPEER needs to match the details from the Windows certificate (from Step 2: Client (Windows): Create certificate).
    Issuer : CN=administrator,O=IBM,C=USA

+ NOTES regarding SSLKEYR:

GSKit adds the ".kdb" suffix for the key database. Do NOT add the kdb extension!
You MUST provide the value for SSLKEYR as follows: full path name of the key store MINUS the .kdb suffix (the .kdb is added at runtime):

Full path name with suffix:	/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb
Full path name minus suffix:	/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS

+ For "2-way authentication".

The default behavior is for the MQ Client application to ask for the authentication of the MQ queue manager, this is called "1-way authentication".

The method of "2-way authentication" is when additionally, the MQ queue manager asks for the authentication of the MQ Client application. This is accomplished by specifying SSLCAUTH(REQUIRED) in the definition of the channel.

Notice that the attribute SSLKEYR is similar to the environment variable MQSSLKEYR in which the SUFFIX .kdb MUST NOT BE SPECIFIED!

Note:
There are 2 channels for 2-way authentication used in this tutorial:
JMSSSL2.SVRCONN for the Windows client, userid "administrator"
JMSSSL2LNX.SVRCONN for the Linux client, userid "mqm"

```
runmqsc QMSTMTLS
  ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS')

  DEFINE CHANNEL('JMSSSL2.SVRCONN') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
  SSLCIPH(TLS_AES_128_GCM_SHA256) SSLCAUTH(REQUIRED) +
  SSLPEER('CN=administrator,O=IBM,C=USA')

  DEFINE CHANNEL('JMSSSL2LNX.SVRCONN') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
  SSLCIPH(TLS_AES_128_GCM_SHA256) SSLCAUTH(REQUIRED) +
  SSLPEER('CN=mqm,O=IBM,C=USA')

  REFRESH SECURITY TYPE(SSL)
  DEFINE QLOCAL(Q1) REPLACE

  END
```

+ For “1-way authentication”:

```
runmqsc QMSTMTLS
  ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS')

  DEFINE CHANNEL('JMSSSL1.SVRCONN') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
  SSLCIPH(TLS_AES_128_GCM_SHA256) SSLCAUTH(OPTIONAL) +
  SSLPEER('')

  REFRESH SECURITY TYPE(SSL)

  DEFINE QLOCAL(Q1) REPLACE

  END
```

++ Step 11: Client (Windows): Add the Linux certificate to the Windows key database

+ Add the public/signer certificate

```
C:\ProgramData\IBM\MQ\ssl>
runmqckm -cert  -add    -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed -label
ibmwebspheremqqmstmtls -file QMSTMTLS.crt  -format ascii
```

```
C:\ProgramData\IBM\MQ\ssl> dir
03/06/2024  12:42 PM          1,170 administrator.crt
03/06/2024  01:05 PM          4,779 key.jks
03/06/2024  12:08 PM            193 key.sth
03/06/2024  12:56 PM          1,156 QMSTMTLS.crt
```

+ List the certificates

```
C:\ProgramData\IBM\MQ\ssl>
runmqckm -cert  -list   -db "C:\ProgramData\IBM\MQ\ssl\key.jks" -stashed
```

```
Certificates in database C:\ProgramData\IBM\MQ\ssl\key.jks:
   ibmwebspheremqadministrator
   dummy
   ibmwebspheremqqmstmtls
```

++ Step 12: Using sample SSLSampleJMS to test the sending of a message from Client (Windows) to Server (Linux)

Go to the directory where the sample is stored.
For this tutorial is:

```
   cd c:\wintools
```

+ 1-way authentication: use channel JMSSSL1.SVRCONN

```
C:\WinTools>java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256
C:\ProgramData\IBM\MQ\ssl\key.jks clientpass
Connecting to:
  Conname = stmichel1.fyre.ibm.com
  Port = 1419
  Channel = JMSSSL1.SVRCONN
  Qmgr = QMSTMTLS
  SSLCiph = TLS_AES_128_GCM_SHA256
  SSLTrustStore = C:\ProgramData\IBM\MQ\ssl\key.jks
  SSLKeyStore = C:\ProgramData\IBM\MQ\ssl\key.jks
  SSLKeyStorePassword = clientpass
Connection Successful!
```

+ 2-way authentication: use channel JMSSSL2.SVRCONN

```
C:\WinTools>java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL2.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256
C:\ProgramData\IBM\MQ\ssl\key.jks clientpass
Connecting to:
  Conname = stmichel1.fyre.ibm.com
  Port = 1419
  Channel = JMSSSL2.SVRCONN
  Qmgr = QMSTMTLS
  SSLCiph = TLS_AES_128_GCM_SHA256
  SSLTrustStore = C:\ProgramData\IBM\MQ\ssl\key.jks
  SSLKeyStore = C:\ProgramData\IBM\MQ\ssl\key.jks
  SSLKeyStorePassword = clientpass
Connection Successful!
```

++ Depending on your environment, you may need to add more parameters to the java invocation.

+ The following includes the -cp attribute:

```
java -cp "C:\Program Files\IBM\MQ\java\lib\com.ibm.mq.allclient.jar;C:\WinTools" -
Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256
C:\ProgramData\IBM\MQ\ssl\key.jks clientpass
```

+ The following includes the -cp attribute, as wells as the javax.net.ssl.keystore and keystorePassord:

```
java -cp "C:\Program Files\IBM\MQ\java\lib\com.ibm.mq.allclient.jar;C:\WinTools" -
Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.trustStorePassword=clientpass -
Djavax.net.ssl.keyStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.keyStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256
C:\ProgramData\IBM\MQ\ssl\key.jks clientpass
```

++ If you want to get the MQ JMS trace and debug ssl trace

If you want to get a MQ JMS trace with 5 files of 400MB each and debug ssl trace you could do the following.

The MQ JMS trace will be written to a "mqjava*.trc" file in the current working directory.
The ssl trace would be written to file "ssl.txt" in the current working directory.

You will need to add the following attributes into the java invocation:

```
-Dcom.ibm.msg.client.commonservices.trace.limit=400000000
-Dcom.ibm.msg.client.commonservices.trace.count=5
-Dcom.ibm.msg.client.commonservices.trace.status=ON
-Djavax.net.debug=ssl:handshake
```

For example:

```
C:\WinTools>java -cp "C:\Program
Files\IBM\MQ\java\lib\com.ibm.mq.allclient.jar;C:\WinTools" -
Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.trustStorePassword=clientpass -
Djavax.net.ssl.keyStore="C:\ProgramData\IBM\MQ\ssl\key.jks" -
Djavax.net.ssl.keyStorePassword=clientpass -
Dcom.ibm.msg.client.commonservices.trace.limit=400000000 -
Dcom.ibm.msg.client.commonservices.trace.count=5 -
Dcom.ibm.msg.client.commonservices.trace.status=ON -
Djavax.net.debug=ssl:handshake SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256
C:\ProgramData\IBM\MQ\ssl\key.jks clientpass 1>ssl.txt 2>&1 &
```

The new files can be shown at the top of the directory listing:

```
C:\WinTools>dir /o-d

C:\WinTools>dir /o-d
03/15/2024  06:44 AM         4,294,229 mqjavaclient_428.trc.0
03/15/2024  06:44 AM            44,985 ssl.txt
03/15/2024  06:44 AM             2,285 mqjms.log.0
```

The file "sst.txt" will contain the print statements inside the sample (in red) and the tracing from javax.net.ssl

```
The IBM MQ messaging client trace mechanism is tracing to
'C:\WinTools\mqjavaclient_428.trc'
Connecting to:
  Conname = stmichel1.fyre.ibm.com
  Port = 1419
  Channel = JMSSSL1.SVRCONN
  Qmgr = QMSTMTLS
  SSLCiph = TLS_AES_128_GCM_SHA256
  SSLTrustStore = C:\ProgramData\IBM\MQ\ssl\key.jks
  SSLKeyStore = C:\ProgramData\IBM\MQ\ssl\key.jks
  SSLKeyStorePassword = clientpass
javax.net.ssl|FINE|01|main|2024-03-15 06:44:25.857 PDT|Thread.java:1178|IBMJSSE2
will not allow protocol SSLv3 per com.ibm.jsse2.disableSSLv3 set to TRUE or default
…
javax.net.ssl|FINE|01|main|2024-03-15 06:44:29.091 PDT|Thread.java:1178|Produced
ClientHello handshake message (
"ClientHello": {
  "client version"      : "TLSv1.2",
  "random"              : "16
…
javax.net.ssl|FINE|0F|RcvThread: com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection@-
919643022[qmid=QMSTMTLS_2024-02-
16_08.25.10,fap=15,channel=JMSSSL1.SVRCONN,ccsid=1208,sharecnv=10,hbint=300,peer=st
michel1.fyre.ibm.com/9.46.98.124(1419),localport=52529,ssl=TLS_AES_128_GCM_SHA256,p
eerDN="CN=QMSTMTLS, O=IBM, C=USA",issuerDN="CN=QMSTMTLS, O=IBM, C=USA"]|2024-03-15
06:44:30.841 PDT|Thread.java:1178|SSLCipher:  Using cipher for decrypt 5
AES/GCM/NoPadding from provider from init IBMJCEPlus version 1.8
Connection Successful!
```

The short file " mqjms.log.0" will contain details on the tracing variables (you can ignore this file)

```
March 15, 2024 6:39:36 AM PDT[main]
com.ibm.msg.client.commonservices.j2se.trace.DefaultTracer
Property 'com.ibm.msg.client.commonservices.trace.status' has value 'ON'
EXPLANATION:
null
ACTION:
Null
```

The file "mqjavaclient_428.trc.0" will contain the MQ JMS tracing.

```
Date: Fri Mar 15 06:39:36 PDT 2024
Process ID:4684
System properties (list may be incomplete due to security policy):
|   awt.toolkit                                    :-  sun.awt.windows.WToolkit
|   com.ibm.cpu.endian                             :-  little
|   com.ibm.fips.home                              :-  C:\Program
Files\IBM\MQ\java\jre
|   com.ibm.fips.mode                              :-  140-2
```

+++ Summary of steps: Client in Linux connecting to a queue manager in Linux

For brevity, only the commands to be used are shown in this section

++ Step 1: Client (Linux): Create SSL client key database, type pkcs12 (jks)

+ Issue "setmqenv" to set the MQ environment variables, including the ones for running Java/JMS applications.

Some ways to issue setmqinst:

- If you have created the "set-mq-inst1" script file mentioned earlier in this document, you can issue:
   . set-mq-inst1

- Issue the command to specify the MQ environment variables:
   . setmqenv -n Installation1

- If necessary, you may need to specify the full path:
   . /opt/mqm/bin/setmqenv -n Installation1

+ Create directory "ssl" (or "tls")

Go to the directory for the MQ Data, such as: cd /var/mqm
cd $MQ_DATA_PATH

Then create subdirectory for “ssl”:
mkdir ssl
cd ssl
You will be at the following directory:
/var/mqm/ssl

+ The following command creates a "jks keystore".

Even though the instances of some commands are shown in 2 lines, it is only a single line.
You must ensure that if you copy/paste the command, you use one single long line:

runmqckm -keydb -create -db /var/mqm/ssl/key.jks -pw clientpass -type pkcs12 -stash

Note:
After stashing the password, for future SSL commands, it is better to use the -stashed command line option than to specify the -pw option.

++ Step 2: Client (Linux) (Skip for 1-way): Create personal certificate

This step is only needed when doing “2-way authentication”.

The step is not needed when doing “1-way authentication” (but it does not hurt if you do it).

+ Create the digital personal certificate.

runmqckm -cert -create -db /var/mqm/ssl/key.jks -stashed -label ibmwebspheremqmqm -dn "CN=mqm,O=IBM,C=USA" -size 2048

Where:
-label is the label name:
      ibmwebspheremqmqm
   It is required to be the concatenation of:
      ibmwebspheremq + userid in lower case
   In this case:
      ibmwebspheremq + mqm
-dn is the "Distinguished Name", notice that for consistency, it is also in lowercase.
-size The recommended size is 2048 bits. The certificates with a size of 1024 are no longer recommended.

+ List the newly created SSL certificate in Windows

runmqckm -cert -list   -db /var/mqm/ssl/key.jks -stashed

```
Certificates in database /var/mqm/ssl/key.jks:
   ibmwebspheremqmqm
   dummy
```

+ List the details of the personal certificate:

runmqckm -cert -details -db /var/mqm/ssl/key.jks -stashed -label ibmwebspheremqmqm

++ Step 3: Client (Linux) (Skip for 1-way): Extract the public SSL client certificate

This step is only needed when doing “2-way authentication”.

But it is not needed when doing “1-way authentication” (but it does not hurt if you do it).

cd /var/mqm/ssl

runmqckm -cert -extract -db /var/mqm/ssl/key.jks -stashed -label ibmwebspheremqmqm -target mqm.crt -format ascii

++ Step 4: Client (Linux) (Skip for 1-way): Copy Linux personal certificate to the SSL server side in Linux

For example, use sftp to transfer the file mqm.crt into /tmp/mqm.crt

++ Step 5: Server (Linux): Create SSL server key database (type CMS)
++ Step 6: Server (Linux): Create certificate
++ Step 7: Server (Linux): Extract the public SSL server certificate

++ Step 8: Server (Linux): Copy Linux certificate to the SSL client side in Linux

For example, use sftp to transfer the file server file QMSTMTLS.crt into /tmp/QMSTMTLS.crt

++ Step 9: Server (Linux) (Skip for 1-way): Add the Linux client certificate to Linux queue manager key database

Ensure that mqm.crt is located in:
/var/mqm/qmgrs/QMSTMTLS/ssl/

```
cd /var/mqm/qmgrs/QMSTMTLS/ssl/
runmqckm -cert  -add     -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed -
label ibmwebspheremqmqm  -file   mqm.crt  -format ascii
```

```
runmqckm -cert  -list    -db "/var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb" -stashed
Certificates in database /var/mqm/qmgrs/QMSTMTLS/ssl/QMSTMTLS.kdb:
   ibmwebspheremqqmstmstls
   ibmwebspheremqadministrator
   ibmwebspheremqmqm
```

++ Step 10: Server (Linux): Run MQSC commands for SSL server side queue manager

++ Step 11: Client (Linux): Add the Linux queue manager certificate to the Linux client key database

```
cd /var/mqm/ssl

runmqckm -cert  -add     -db /var/mqm/ssl/key.jks -stashed -label
ibmwebspheremqqmstmtls -file QMSTMTLS.crt  -format ascii
```

```
runmqckm -cert  -list    -db /var/mqm/ssl/key.jks -stashed
Certificates in database /var/mqm/ssl/key.jks:
   ibmwebspheremqmqm
   dummy
   ibmwebspheremqqmstmtls
```

++ Step 12: Using sample SSLSampleJMS to test the sending of a message from Client (Linux) to Server (Linux)

1-way authentication:

```
mqm@c46968v1.fyre.ibm.com: /home/mqm/ssl
$ java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks clientpass
Connecting to:
  Conname = stmichel1.fyre.ibm.com
  Port = 1419
  Channel = JMSSSL1.SVRCONN
  Qmgr = QMSTMTLS
  SSLCiph = TLS_AES_128_GCM_SHA256
  SSLTrustStore = /var/mqm/ssl/key.jks
  SSLKeyStore = /var/mqm/ssl/key.jks
  SSLKeyStorePassword = clientpass
Connection Successful!
```

2-way authentication:

```
mqm@c46968v1.fyre.ibm.com: /home/mqm/ssl
$ java -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL2LNX.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks
clientpass
  Conname = stmichel1.fyre.ibm.com
  Port = 1419
  Channel = JMSSSL2LNX.SVRCONN
  Qmgr = QMSTMTLS
  SSLCiph = TLS_AES_128_GCM_SHA256
  SSLTrustStore = /var/mqm/ssl/key.jks
  SSLKeyStore = /var/mqm/ssl/key.jks
  SSLKeyStorePassword = clientpass
Connection Successful!
```

++ Depending on your environment, you may need to add more parameters to the java invocation.

+ The following includes the -cp attribute:

java -cp /opt/mqm/java/lib/com.ibm.mq.allclient.jar:/home/mqm/ssl -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419 JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks clientpass

+ The following includes the -cp attribute, as wells as the javax.net.ssl.keystore and keystorePassord:

java -cp /opt/mqm/java/lib/com.ibm.mq.allclient.jar:/home/mqm/ssl -Dcom.ibm.mq.cfg.useIBMCipherMappings=false -Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -Djavax.net.ssl.trustStorePassword=clientpass -Djavax.net.ssl.keyStore=/var/mqm/ssl/key.jks -Djavax.net.ssl.keyStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419 JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks clientpass

++ If you want to get the MQ JMS trace and debug ssl trace

If you want to get a MQ JMS trace with 5 files of 400MB each and debug ssl trace you could do the following.

The MQ JMS trace will be written to a "mqjava*.trc" file in the current working directory.
The ssl trace would be written to file "ssl.txt" in the current working directory.

You will need to add the following attributes into the java invocation:

```
-Dcom.ibm.msg.client.commonservices.trace.limit=400000000
-Dcom.ibm.msg.client.commonservices.trace.count=5
-Dcom.ibm.msg.client.commonservices.trace.status=ON
-Djavax.net.debug=ssl:handshake
```

For example:

```
mqm@c46968v1.fyre.ibm.com: /home/mqm/ssl
$ java -cp /opt/mqm/java/lib/com.ibm.mq.allclient.jar:/home/mqm/ssl  -
Dcom.ibm.mq.cfg.useIBMCipherMappings=false -
Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -
Djavax.net.ssl.trustStorePassword=clientpass -
Djavax.net.ssl.keyStore=/var/mqm/ssl/key.jks -
Djavax.net.ssl.keyStorePassword=clientpass -
Dcom.ibm.msg.client.commonservices.trace.limit=400000000 -
Dcom.ibm.msg.client.commonservices.trace.count=5 -
Dcom.ibm.msg.client.commonservices.trace.status=ON -
Djavax.net.debug=ssl:handshake SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks
clientpass 1>ssl.txt 2>&1 &
```

The new files can be shown at the top of the directory listing:

```
mqm@c46968v1.fyre.ibm.com: /home/mqm/ssl
$ ls -lt
-rw-rw-r-- 1 mqm mqm 4083943 Mar 15 07:00 mqjavaclient_104180.trc.0
-rw-rw-r-- 1 mqm mqm   16917 Mar 15 07:00 ssl.txt
-rw-rw-r-- 1 mqm mqm    4443 Mar 15 07:00 mqjms.log.0
```

The file "sst.txt" will contain the print statements inside the sample (in red) and the tracing from javax.net.ssl

```
The IBM MQ messaging client trace mechanism is tracing to
'/home/mqm/ssl/mqjavaclient_104180.trc'
Connecting to:
  Conname = stmichel1.fyre.ibm.com
  Port = 1419
  Channel = JMSSSL1.SVRCONN
  Qmgr = QMSTMTLS
…
javax.net.ssl|FINE|01|main|2024-03-15 06:44:25.857 PDT|Thread.java:1178|IBMJSSE2
will not allow protocol SSLv3 per com.ibm.jsse2.disableSSLv3 set to TRUE or default
…
javax.net.ssl|FINE|01|main|2024-03-15 06:44:29.091 PDT|Thread.java:1178|Produced
ClientHello handshake message (
"ClientHello": {
  "client version"      : "TLSv1.2",
  "random"              : "16
…
javax.net.ssl|FINE|0F|RcvThread: com.ibm.mq.jmqi.remote.impl.RemoteTCPConnection@-
919643022[qmid=QMSTMTLS_2024-02-
16_08.25.10,fap=15,channel=JMSSSL1.SVRCONN,ccsid=1208,sharecnv=10,hbint=300,peer=st
michel1.fyre.ibm.com/9.46.98.124(1419),localport=52529,ssl=TLS_AES_128_GCM_SHA256,p
eerDN="CN=QMSTMTLS, O=IBM, C=USA",issuerDN="CN=QMSTMTLS, O=IBM, C=USA"]|2024-03-15
06:44:30.841 PDT|Thread.java:1178|SSLCipher:  Using cipher for decrypt 5
AES/GCM/NoPadding from provider from init IBMJCEPlus version 1.8
Connection Successful!
```

The short file " mqjms.log.0" will contain details on the tracing variables (you can ignore this file)

```
March 15, 2024 6:59:02 AM PDT[main]
com.ibm.msg.client.commonservices.j2se.trace.DefaultTracer
Property 'com.ibm.msg.client.commonservices.trace.status' has value 'ON'
EXPLANATION:
null
ACTION:
Null
```

The file " mqjavaclient_104180.trc.0" will contain the MQ JMS tracing.

```
Date: Fri Mar 15 07:00:35 PDT 2024
Process ID:104180
System properties (list may be incomplete due to security policy):
|   awt.toolkit                                    :-  sun.awt.X11.XToolkit
|   com.ibm.mq.cfg.useIBMCipherMappings            :-  false
|   com.ibm.msg.client.commonservices.trace.count  :-  5
|   com.ibm.msg.client.commonservices.trace.limit  :-  400000000
```

++ Troubleshooting scenario

Problem:

You execute the following command and do not specify any -D parameters.

```
mqm@c46968v1.fyre.ibm.com: /home/mqm/ssl
$ java SSLSampleJMS stmichel1.fyre.ibm.com 1419 JMSSSL1.SVRCONN QMSTMTLS
TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks clientpass
Connecting to:
  Conname = stmichel1.fyre.ibm.com
  Port = 1419
  Channel = JMSSSL1.SVRCONN
  Qmgr = QMSTMTLS
  SSLCiph = TLS_AES_128_GCM_SHA256
  SSLTrustStore = /var/mqm/ssl/key.jks
  SSLKeyStore = /var/mqm/ssl/key.jks
  SSLKeyStorePassword = clientpass
com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ0018: Failed to connect to queue
manager 'QMSTMTLS' with connection mode 'Client' and host name
'stmichel1.fyre.ibm.com(1419)'.
Check the queue manager is started and if running in client mode, check there is a listener
running. Please see the linked exception for more information.
…
        at SSLSampleJMS.main(SSLSampleJMS.java:136)
Caused by: com.ibm.mq.MQException: JMSCMQ0001: IBM MQ call failed with compcode '2'
('MQCC_FAILED') reason '2397' ('MQRC_JSSE_ERROR').
        at
com.ibm.msg.client.wmq.common.internal.Reason.createException(Reason.java:203)
        ... 9 more
Caused by: com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9204: Connection to host
'stmichel1.fyre.ibm.com(1419)' rejected.
[1=com.ibm.mq.jmqi.JmqiException[CC=2;RC=2397;AMQ9771: SSL handshake failed.
[1=javax.net.ssl.SSLException[Unexpected error:
java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be
non-empty],3=stmichel1.fyre.ibm.com/9.46.98.124:1419
(stmichel1.fyre.ibm.com),4=SSLSocket.startHandshake,5=default]],3=stmichel1.fyre.ibm.co
m(1419),4=,5=RemoteTCPConnection.protocolConnect]
        at
com.ibm.mq.jmqi.remote.api.RemoteFAP$Connector.jmqiConnect(RemoteFAP.java:13687)
…
Caused by: com.ibm.mq.jmqi.JmqiException: CC=2;RC=2397;AMQ9771: SSL handshake
failed. [1=javax.net.ssl.SSLException[Unexpected error:
java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be
non-empty],3=stmichel1.fyre.ibm.com/9.46.98.124:1419
(stmichel1.fyre.ibm.com),4=SSLSocket.startHandshake,5=default]
        at …
```

```
Caused by: javax.net.ssl.SSLException: Unexpected error:
java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be
non-empty
	at sun.security.ssl.Alert.createSSLException(Alert.java:133)
…
Caused by: java.lang.RuntimeException: Unexpected error:
java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be
non-empty
	at sun.security.validator.PKIXValidator.<init>(PKIXValidator.java:104)
…
Caused by: java.security.InvalidAlgorithmParameterException: the trustAnchors parameter
must be non-empty
	at java.security.cert.PKIXParameters.setTrustAnchors(PKIXParameters.java:200)
```

Solution:
You need to specify the "trustStore" parameters during the java invocation.

```
java -Djavax.net.ssl.trustStore=/var/mqm/ssl/key.jks -
Djavax.net.ssl.trustStorePassword=clientpass SSLSampleJMS stmichel1.fyre.ibm.com 1419
JMSSSL1.SVRCONN QMSTMTLS TLS_AES_128_GCM_SHA256 /var/mqm/ssl/key.jks clientpass
```

+++ end +++