Configuration and basic test of remote clients for IBM MQ 9.1, 9.2 and 9.3 Advanced Message Security (AMS) in Linux

https://www.ibm.com/support/pages/node/6244608

Date last updated: 07-Jun-2023

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

+++ Thanks to Bob Gibson for his suggestions for improving this tutorial!

+ Update on 07-Jun-2023:
- MQ 9.3.2 CD and MQ 9.3.0.2 LTS under RHEL 8.6 were used to validate the scenarios.
- Minor corrections and improvements were done (Thank you Bob Gibson!)

+ Update on 30-Apr-2022:
- MQ 9.2.5 CD was used under RHEL 8.5 to validate the scenarios.
- Minor corrections and improvements were done (Thank you Bob Gibson!)
- Only the users who are going to put/get messages from an AMS protected queue need to create keystore and certificates.
The queue manager does NOT use the certificates from these users.

+++ Objective

The objective of this technical document is to describe in detail how to configure the environment for two MQ users who will be using remote access to a queue manager that is using the MQ Advanced Message Security (AMS).

This document shows how to perform a basic test using the following MQ samples which use remote network or client connection: amqsputc and amqsgetc by 3 users in 3 hosts:

- host-1 for the queue manager (QM_VERIFY_AMS in port 1456).
  User ‘fulano’ will not be authorized to put/get. It is going to be used for comparison.

- host-2 for user ‘mary’ who will be authorized to put a message into a protected queue by AMS.

- host-3 for user ‘john’ who will be authorized to get a message from the protected queue.

Because the use or lack of use of encryption on the server-connection channel is not relevant for AMS (AMS protects messages at rest at the queue manager), then, in order to keep this tutorial short, the server-connection channel that is going to be used is NOT enabled for TLS.

However, the userid and password will be used with MQ samples amqsputc and amqsgetc.
For the installation, configuration of AMS and initial testing of put/get using **"bindings mode"** (within the same host as the queue manager) see the following tutorial:

https://www.ibm.com/support/pages/node/598373
Installation, Configuration and Basic Test of MQ 9.0 Advanced Message Security (AMS) in Linux

The chapters in this techdoc are:

Chapter 1: Topology and Configuration
Chapter 2: Creating key database and certificates, host-2 (mary) / host-3 (john)
Chapter 3: Creating keystore.conf for each user
Chapter 4: Sharing Certificates
Chapter 5: Using amqsputc from host-2 and amqsgetc from host-3

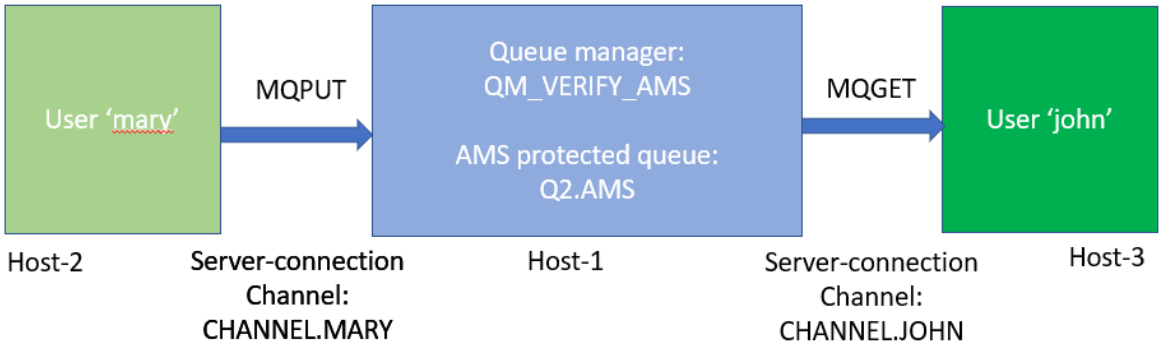
++
+++ Chapter 1: Topology and Configuration
++

The remote users are NOT MQ-administrators, therefore, when the accounts are created, they should not belong to the group “mqm” in host-1 (the one that has the queue manager).

+++ Topology

This is the topology used for this tutorial:
3 different hosts (2 remote users)

The arrows show the message flow.



Each user will have a dedicated server-connection channel.
The main reason is that this can provide flexibility regarding the lack of encryption (by default) of the channel, or the enablement of encryption, if desired.

In this tutorial, the initial scenario is without using encryption in the server-connection channels.

++ Test recommendation: to have 3 separate command prompt windows

Because this scenario describes the tasks done by multiple users, it is best to create at least three (3) separate command prompt windows, which helps to reduce confusion.

Window 1 for host-1: for users “root” and “mqm”
Window 2 for host-2: for user “mary”, amqsputc
Window 3 for host-3: for user “john”, amqsgetc

+++ Configuration

See the initial chapters for the related tutorial mentioned in page 2 for installing, creating the queue manager, creating the local group and local users, for creating the policy, etc.

++ host-1: queue manager

Hostname: volterra1.fyre.ibm.com
OS: $ **cat /etc/system-release**
Red Hat Enterprise Linux release 8.6 (Ootp
MQ: $ **dspmqver -f2**
Version: 9.3.0.5

+ Login as user “root”.

Install the MQ server code.

Use line commands or the GUI or another administrative tool to create:

Group: mquser
groupadd -g 1005 mquser

Users:
fulano:
useradd -u 1021 -g mquser -s /bin/bash -d /home/fulano -m fulano
mary:
useradd -u 1033 -g mquser -s /bin/bash -d /home/mary -m mary
john:
useradd -u 1034 -g mquser -s /bin/bash -d /home/john -m john

+ Login as user “mqm”

- Configuration

Queue Manager:
QM_VERIFY_AMS (Port 1456)

Protected queue:
Q2.AMS

Server-connection channels :
CHANNEL.MARY (defined with no encryption)
CHANNEL.JOHN (defined with no encryption)

-Establish the environment variables for MQ

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

-Create the queue manager.

$ crtmqm -u DLQ -p 1456 QM_VERIFY_AMS

The -u flag indicates which queue is going to be the dead letter queue (DLQ).
Hint: Many MQ Explorer users hide the SYSTEM* queues and thus, if you use the SYSTEM.DEAD.LETTER.QUEUE as the DLQ, then it will be hidden and you might not notice if there are messages in the dead letter queue

-Start the queue manager

$ strmqm QM_VERIFY_AMS

-Configure the queue manager

$ runmqsc QM_VERIFY_AMS

Define a normal queue which will NOT be protected by AMS

define qlocal(Q1)

Define the testing queue which will be protected by AMS

define qlocal(Q2.AMS)

Define a server-connection channel to be used by a remote MQ Explorer

define channel(SYSTEM.ADMIN.SVRCONN) chltype(SVRCONN)

Define the DLQ

define qlocal(DLQ) like(SYSTEM.DEAD.LETTER.QUEUE)

For MQ 7.1 and later and if desiring to allow remote connections by
an MQ Administrator (to avoid return code 2035). This is OK for test queue managers.
This security feature does NOT interfere at all with AMS.

set CHLAUTH(*) TYPE(BLOCKUSER) USERLIST('nobody','*MQADMIN')
set CHLAUTH(SYSTEM.ADMIN.*) TYPE(BLOCKUSER) USERLIST('nobody')

For MQ 8.0 and later to disable password for remote MQ administrators.
This security feature does NOT interfere at all with AMS.

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) CHCKCLNT(OPTIONAL)

REFRESH SECURITY TYPE(CONNAUTH)

Define the server-connection channel for the remote access for mary
For now, let’s not use encryption.

DEFINE CHANNEL(CHANNEL.MARY) CHLTYPE(SVRCONN)

At this point, the channel is open to anyone! Everything gets thru!

Define the 2 channel authentication records for user mary to be able to use
this server-connection channel
There are 2 rules that need to be defined.
Notice that the 1st rule is called the "back stop" rule, which blocks the access to all users for that channel (it is like a brick wall with no holes). Just by itself, this rule does not allow the channel to be usable.
What makes the channel usable is the 2nd rule, which opens a hole in the wall established by the 1st rule.

1st rule - back stop - blocks all users (it is like a wall)
At this point, the channel is NOT usable at all! Nothing gets thru!

SET CHLAUTH(CHANNEL.MARY) TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)

2nd rule - allows one userid to connect (it is like a hole in the wall)

SET CHLAUTH(CHANNEL.MARY) TYPE(USERMAP) CLNTUSER('mary') USERSRC(MAP) +
MCAUSER('mary') ADDRESS('*') ACTION(ADD)

Define the server-connection channels for the remote access for john and add the channel authentication records.
For now, let's not use encryption.

DEFINE CHANNEL(CHANNEL.JOHN) CHLTYPE(SVRCONN)

SET CHLAUTH(CHANNEL.JOHN) TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)

SET CHLAUTH(CHANNEL.JOHN) TYPE(USERMAP) CLNTUSER('john') USERSRC(MAP) +
MCAUSER('john') ADDRESS('*') ACTION(ADD)

exit runmqsc

end

+ Authorizing the users

The following commands were used to authorize the users to connect to the queue manager:

setmqaut -m QM_VERIFY_AMS -t qmgr -p mary +connect +inq +dsp
setmqaut -m QM_VERIFY_AMS -t qmgr -p john +connect +inq +dsp

Work with the queue Q1: mary and john can put and get.

setmqaut -m QM_VERIFY_AMS -n Q1 -t queue -p mary +get +put +browse +dsp
setmqaut -m QM_VERIFY_AMS -n Q1 -t queue -p john +get +put +browse +dsp

Work with the queue Q2.AMS: mary can put and john can get.

setmqaut -m QM_VERIFY_AMS -n Q2.AMS -t queue -p mary +put +browse +dsp
setmqaut -m QM_VERIFY_AMS -n Q2.AMS -t queue -p john +get +browse +dsp

Additionally, it is necessary to allow the two users to browse the AMS system policy queue, and put messages on the AMS error queue.

setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p mary +browse
setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p john +browse

setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p mary +put
setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p john +put

+ Define AMS policy

Define the AMS policy to protect Queue Q2.AMS using:
  signature algorithm SIGNALG(SHA256),
  encryption algorithm ENCALG(AES256) and
  key reuse KEYREUSE(DISABLED)
… and to allow users to:
  User mary to put (the signer or SIGNER)
  User john to get (the recipient or RECIP)

+ Line command:

$ setmqspl -m QM_VERIFY_AMS -p Q2.AMS -s SHA256 -a "CN=mary,O=IBM,C=US" -e AES256 -r "CN=john,O=IBM,C=US" -c 0

$ **dspmqspl -m QM_VERIFY_AMS -p Q2.AMS**
Policy Details:
Policy name: Q2.AMS
Quality of protection: PRIVACY
Signature algorithm: SHA256
Encryption algorithm: AES256
Signer DNs:
  CN=mary,O=IBM,C=US
Recipient DNs:
  CN=john,O=IBM,C=US
Key reuse count: 0
Toleration: 0

To get the syntax similar to setmqspl, specify the flag: -export

$ **dspmqspl -m QM_VERIFY_AMS -p Q2.AMS -export**
setmqspl -m QM_VERIFY_AMS -p Q2.AMS -s SHA256 -a "CN=mary,O=IBM,C=US" -e AES256 -r "CN=john,O=IBM,C=US" -c 0 -t 0

+ Using runmqsc.

For completeness, in case that you want to use runmqsc instead of a line command, here are the details.

Note that you have to use single quotes for the distinguished names in SIGNER and RECIP.

```
SET POLICY(Q2.AMS) SIGNALG(SHA256) ENCALG(AES256) KEYREUSE(DISABLED) +
   ENFORCE  SIGNER('CN=mary,O=IBM,C=US') RECIP('CN=john,O=IBM,C=US')


DISPLAY POLICY(Q2.AMS)
AMQ9086I: Display IBM MQ Advanced Message Security policy details.
   POLICY(Q2.AMS)                  SIGNALG(SHA256)
   ENCALG(AES256)                  SIGNER(CN=mary,O=IBM,C=US)
   RECIP(CN=john,O=IBM,C=US)       KEYREUSE(DISABLED)
   ENFORCE
```

++ Host-2: has user ‘mary’ who is authorized to put messages

As root install MQ.

As root create a group and a user:

Group: mquser
groupadd -g 1005 mquser

User: mary
useradd -u 1033 -g mquser -s /bin/bash -d /home/mary -m mary

Setup the password for this user:
passwd mary

++ Host-3 has user ‘john’ who is authorized to get messages

As root install MQ.

As root create a group and a user:

Group: mquser
groupadd -g 1005 mquser

Users: john
useradd -u 1034 -g mquser -s /bin/bash -d /home/john -m john

Setup the password for this user:
passwd john

++
+++ Chapter 2: Creating key database and certificates, host-2 (mary) / host-3 (john)
++

To encrypt the message, the AMS interceptors require the public key of the users. Thus, the key database of user identities mapped to public and private keys must be created.

In this scenario, we are using self-signed certificates which can be created without using a Certificate Authority.
For production systems, it is advisable not to use self-signed certificates however instead rely on certificates signed by a Certificate Authority.

For CMS repositories, the file extension MUST be .kdb and the password MUST be stashed.

The C sample programs (i.e., amqsputc & amqsgetc) must have a CMS keystore repository.

+ Host-2: User mary

Login as user mary.

The umask used in this example is the following:

umask
0022

Note: This umask is used by the operating system to setup the permissions when creating files. The following is an example in which a file is created with
644 (rw-r--r--) file permissions:
$ touch file.txt
$ ls -l file.txt
-rw-r--r--1 mary mquser 0 Apr 22 10:52 file.txt

Create a directory to store the GSKit related files, including the key database.
mkdir /home/mary/ssl

Create a new key database for user mary:
runmqakm -keydb -create -db /home/mary/ssl/marykey.kdb -pw passw0rd -type cms -stash

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

Create a self-signed certificate identifying the user mary for use in encryption

Option 1: Specifying -stashed
runmqakm -cert -create -db /home/mary/ssl/marykey.kdb -stashed
-label Mary_Cert -dn "CN=mary,O=IBM,C=US" -default_cert yes

Option 2: Specifying -pw passwordPhrase
runmqakm -cert -create -db /home/mary/ssl/marykey.kdb -pw passw0rd
-label Mary_Cert -dn "CN=mary,O=IBM,C=US" -default_cert yes

Notes:
- The 'label' parameter specifies the name for the certificate, which interceptors will look up to receive necessary information.
- The 'dn' parameter specifies the details of the Distinguished Name (DN), which must be unique for each user.\
- After stashing the kdb password, it is better to use the -stashed command line option than to specify the -pw option.

+ Host-3: User john

Login as user john.

The umask used in this example is:
umask
0022

Create a directory to store the GSKit related files, including the key database.
mkdir /home/john/ssl

Create a new key database for the user john

runmqakm -keydb -create -db /home/john/ssl/johnkey.kdb -pw passw0rd -type cms -stash

Create a self-signed certificate identifying the user john for use in encryption

Option 1: Specifying -stashed
runmqakm -cert -create -db /home/john/ssl/johnkey.kdb -stashed
-label John_Cert -dn "CN=john,O=IBM,C=US" -default_cert yes

Option 2: Specifying -pw passwordPhrase
runmqakm -cert -create -db /home/john/ssl/johnkey.kdb -pw passw0rd
-label John_Cert -dn "CN=john,O=IBM,C=US" -default_cert yes

++
+++ Chapter 3: Creating keystore.conf for each user
++

You must point MQ Advanced Message Security interceptors to the directory where the key databases and certificates are located.
This is done via the keystore.conf file, which hold that information in the plain text form.

Each user must have a separate keystore.conf file.
Therefore, this step should be done for both mary and john.

The content of keystore.conf must be of the form:

cms.keystore = <dir>/keystore_file
cms.certificate = certificate_label

Notes:

- The path to the keystore file must be provided with no file extension.

- $HOME/.mqs/keystore.conf is the default location where MQ AMS searches for the keystore.conf file.

- The default location of the keystore.conf file is:
[AIX][IBM i][Linux]On IBM i, AIX and Linux: $HOME/.mqs/keystore.conf
[Windows]On Windows: %HOMEDRIVE%%HOMEPATH%\.mqs\keystore.conf

- If you are using a specified keystore filename and location, you should use the following commands
For Java™: java -DMQS_KEYSTORE_CONF=path/filename app_name
For C Client and Server:
On AIX and Linux: **export MQS_KEYSTORE_CONF=path/filename**
On Windows: set MQS_KEYSTORE_CONF=path\filename

- If the keystore.conf file does not contain a cms.keystore attribute, an error 2035 NOT_AUTHORIZED will result.

```
$ amqsputc SECURE.Q AMS.QM
Sample AMQSPUT0 start
Enter password: ********
target queue is SECURE.Q
MQOPEN ended with reason code 2035
unable to open queue for output
Sample AMQSPUT0 end
```

- A message like this will be written to the $MQ_DATA_PATH/errors/AMQERR01.LOG file:

```
05/25/2023 03:30:00 PM - Process(278350.1) User(alice) Program(amqsputc)
                    Host(armorer1.fyre.ibm.com) Installation(Installation1)
                    VRMF(9.2.5.0)

AMQ9063E: The IBM MQ security policy interceptor could not parse the keystore
configuration file.


EXPLANATION:
The IBM MQ security policy interceptor could not parse the 'cms' attribute key
in the keystore configuration file '/home/mary/ssl/keystore.conf'.
ACTION:
Make sure the keystore configuration file contains all required attributes and
does not contain any duplicate attribute keys.
```

+ Host-2: User mary

Add into your .bashrc file the following variable to specify the location nof the conf file:
export MQS_KEYSTORE_CONF=$HOME/ssl/keystore.conf

Create file:
vi $HOME/ssl/keystore.conf

The contents is:

```
cms.keystore = /home/mary/ssl/marykey
cms.certificate = Mary_Cert
```

+ Host-3: User john

Add into your .bashrc file the following variable to specify the location nof the conf file:
export MQS_KEYSTORE_CONF=$HOME/ssl/keystore.conf

Create file:
vi $HOME/ssl/keystore.conf

The contents is:

cms.keystore = /home/john/ssl/johnkey
cms.certificate = John_Cert

++
+++ Chapter 4: Sharing Certificates
++

It is necessary to share the certificates between the two key databases so that each user can successfully identify each other.
Because the users are located in different hosts, if you are using ftp, then specify the file transfer as “ascii” because the certificate is in plain text.

+ Window 2: User mary

Export the certificate identifying mary to a file.
In this case, for convenience, the file will be located in $HOME/ssl, the same directory as the keystore.
The resulting file will be written as ascii text, which is the default (-format ascii).

runmqakm -cert -extract -db $HOME/ssl/marykey.kdb -stashed
-label Mary_Cert -target $HOME/ssl/mary_public.arm

Copy/transfer the file **$HOME/ssl/mary_public.arm** to host-3 as “ascii”/”text” file.
The file will be stored in the directory "ssl" for user john in host-3.

+ Host-3: User john

Verify that the file **/home/john/ssl/mary_public.arm** was transferred from host-2 to host-3.

Add the certificate from mary into john's keystore:

runmqakm -cert -add -db $HOME/ssl/johnkey.kdb -stashed -label Mary_Cert
-file $HOME/ssl/mary_public.arm

Print the details of the certificate for mary, to verify that it is indeed in the keystore,
runmqakm -cert -details -db $HOME/ssl/johnkey.kdb -stashed -label Mary_Cert

+ begin excerpt

Label : Mary_Cert
Key Size : 2048
Version : X509 V3
Serial : 4cf3ba34081bb9a5
Issuer : CN=mary,O=IBM,C=US
Subject : CN=mary,O=IBM,C=US

+ end excerpt

Export the certificate identifying john to a file.
In this case, for convenience, the file will be located in $HOME/ssl, the same directory as the keystore.

runmqakm -cert -extract -db $HOME/ssl/johnkey.kdb -stashed -label John_Cert -target $HOME/ssl/john_public.arm

Copy/transfer the file **john_public.arm** to host-2 as “ascii”/”text” file.

+ Host-2: User mary

Back to Host-2 for mary.

Verify that the file **/home/mary/ssl/john_public.arm** was transferred from host-3 to host-2.

Add the certificate for john to mary's keystore:

runmqakm -cert -add -db $HOME/ssl/marykey.kdb -stashed -label John_Cert -file $HOME/ssl/john_public.arm

Print the details for the certificate for john.

runmqakm -cert -details -db $HOME/ssl/marykey.kdb -stashed -label John_Cert

(Similar results as for mary)

+ begin excerpt

Label : John_Cert
Key Size : 2048
Version : X509 V3
Serial : 730ea7a7942e8d52
Issuer : CN=john,O=IBM,C=US
Subject : CN=john,O=IBM,C=US

+ end excerpt

++
+++ Chapter 5: Using amqsputc from host-2 and amqsgetc from host-3
++

Let's test the setup by putting a message as user mary from host-2 and reading the message as user john from host-3

+ Host-2: User mary

It is necessary to indicate to the MQ sample amqsputc how to reach the queue manager.

Let’s try the simpler method which is to use the MQSERVER environment variable (which can only be used for plaintext, because it does not support TLS transmission).

mary@riggioni1.fyre.ibm.com: /home/mary
$ **export MQSERVER='CHANNEL.MARY/TCP/volterra1.fyre.ibm.com(1456)'**

The MQ samples use an environment variable called MQ_SAMP_USERID to specify the user name and then the password will be requested when running the sample.

mary@riggioni1.fyre.ibm.com: /home/mary
$ **export MQSAMP_USER_ID=mary**

As user mary, put a message using a sample application.
First, you will need to enter the password for user ‘mary’ (must the password for that user in host-1, the host of the queue manager).
Then type the text of the message, and press Enter.

$ amqsputc Q2.AMS QM_VERIFY_AMS
amqsputc Q2.AMS QM_VERIFY_AMS
Sample AMQSPUT0 start
Enter password: **********
target queue is Q2.AMS
putting a message from a remote connection into an AMS protected queue

Sample AMQSPUT0 end

+ Host-3: User john

It is necessary to indicate to the MQ sample amqsputc, how to reach the queue manager.

john@suvereto11.fyre.ibm.com: /home/john
$ **export MQSERVER='CHANNEL.JOHN/TCP/volterra1.fyre.ibm.com(1456)'**
The MQ samples use an environment variable called MQ_SAMP_USERID to specify the user name and then the password will be requested when running the sample.

john@suvereto1.fyre.ibm.com: /home/john
$ **export MQSAMP_USER_ID=john**

As user john, get a message using a sample application.
You will need to enter the password for user 'john' (must the password for that user in host-1, the host of the queue manager).

john@suvereto1.fyre.ibm.com: /home/john
$ **amqsgetc Q2.AMS QM_VERIFY_AMS**
Sample AMQSGET0 start
Enter password: **********
message <putting a message from a remote connection into an AMS protected queue>

no more messages
Sample AMQSGET0 end

Conclusion:
User 'mary' was able to put a message into a queue that is protected by AMS and user 'john' was able to read it. Yeah!

+++ end +++