

IBM z/OS Connect EE V3.0

# Customization - Security and MQ



*Lab Version Date: July 25, 2020*

## Table of Contents

<b>Overview .....</b>	<b>4</b>
<b><i>Enabling Identity Propagation to MQ .....</i></b>	<b>5</b>
<i>Test with identity propagation not enabled.....</i>	<i>5</i>
<i>Summary .....</i>	<i>8</i>
<b><i>Enabling TLS security to a MQ queue Manager from a zCEE Server .....</i></b>	<b>9</b>
<i>Creating MQ SAF resources.....</i>	<i>9</i>
<i>Enabling MQ TLS support for a queue manager .....</i>	<i>11</i>
<i>Summary .....</i>	<i>15</i>

**Important:** On the desktop there is a file named *Security CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

## ***General Exercise Information and Guidelines***

- ✓ This exercise requires the completion of the *zCEE Basic Configuration* and *zCEE Basic Security Configurations* exercises before it can be performed.
- ✓ This exercise requires using z/OS user identities *FRED* and *USER1*. The password for these users will be provided by the lab instructions.
- ✓ There are examples of *server.xml* scattered through this exercise. Your *server.xml* may differ depending on which exercises have been previously performed. Be sure the red lines in these examples are either added or already present.
- ✓ The acronyms RACF (resource access control facility) and SAF (system authorization facility) are used in this exercise. RACF is the IBM security manager product whereas SAF is a generic term for any security manager product, e.g. ACF2 or Top Secret or RACF. An attempt has been to use SAF when referring to information appropriate for any SAF product and to use RACF when referring to specific RACF commands or examples.
- ✓ Any time you have any questions about the use of IBM z/OS Explorer, 3270 screens, features or tools, do not hesitate to ask the instructor for assistance.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *Security CopyPaste* file on the desktop.
- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see when performing this exercise. These differences should not impact the completion of this exercise.

## Overview

This exercise demonstrates use examples to show the steps required to enable security between a z/OS Connect EE (zCEE) server and a MQ queue manager.

In part one of the exercise, identity propagation between a z/OS Connect EE (zCEE) server and the queue manger is enabled and tested. Identity propagation in this context means that the identity authenticated by the zCEE server will be sent to the queue manager for subsequent MQ authorization checks.

In part two of the exercise, TLS support will be added to the MQ queue manger and the steps required to enable the exchange of certificates for mutual authentication between the zCEE server and the MQ queue manger will be enabled. Finally, TLS security and mutual authentication will be demonstrated.

## Enabling Identity Propagation to MQ

### Test with identity propagation not enabled

- \_\_\_ 1. Edit the *server.xml* configuration file for the *myServer* server, e.g. */var/zosconnect/servers/myServer/server.xml* and add includes for *shared.xml* and *mqClient.xml*, see below:

```
<include location="${server.config.dir}/includes/mqClient.xml"/>
```

```
<include location="${server.config.dir}/includes/mqClient.xml"/>
```

```
<include location="${server.config.dir}/includes/safSecurity.xml"/>
<include location="${server.config.dir}/includes/ipic.xml"/>
<include location="${server.config.dir}/includes/keyringMutual.xml"/>
<include location="${server.config.dir}/includes/groupAccess.xml"/>
<include location="${server.config.dir}/includes/shared.xml"/>
<include location="${server.config.dir}/includes/mqClient.xml"/>
```

The include for “shared.xml” may already be present, depending on whether other security exercises have been performed.

- \_\_\_ 2. Issue the MVS commands **P BAQSTRT** to stop the z/OS Connect server and then enter MVS command **S BAQSTRT** to restart the server.

**Tech-Tip:** MVS and JES2 commands can be entered from SDSF by enter a / (slash) on the command line followed by the command itself (e.g. /D T). The command results can be found in the system log. If a command is especially long, then enter a / (slash) to display a *SDSF – System Command Extension* panel where a command can span multiple lines. When an MVS command must be entered, the instructions in these exercises will indicate that the command is an MVS command and you may enter the command at the prompt by using the / (slash) prefix or using the *SDSF – System Command Extension* panel.

- \_\_\_ 3. Open a DOS command prompt and go to directory *c:/z/admin*.
- \_\_\_ 4. Enter the cURL command below:

```
curl -X POST -w '%{http_code}' --cacert certauth.pem --cert fred.p12:secret --cert-type P12 --
header "Content-Type: application/json" --data @MQMessage.json
https://wg31.washington.ibm.com:9443/mqapi/queue
```

```
c:\z\admin>curl -X POST -w '%{http_code}' --cacert certauth.pem --cert
fred.p12:secret --cert-type P12 --header "Content-Type: application/json" --data
@MQMessage.json https://wg31.washington.ibm.com:9443/mqapi/queue
'204'
```

Invoking this API will cause contents of the JSON message in file *MQMessage.json* to be placed on the queue *ZCEE.DEFAULT.DEFAULT.QUEUE* in queue manager *ZMQ1*. The “204” response indicates that the server has successfully fulfilled the request and there was no response message returned, which is normal for a MQ put REST request. Which queue manager and which queue in that queue manager was determined by the JNDI names in the service’s meta data (see below).

The screenshot shows the 'Service Project Editor: Configuration' window for the 'mqPutService' service. Under the 'Required Configuration' section, the following fields are visible:

- Connection factory JNDI name:
- Destination JNDI name:
- Coded character set identifier (CCSID):
- MQMD format:
- Is message persistent: ☐
- Expiry:

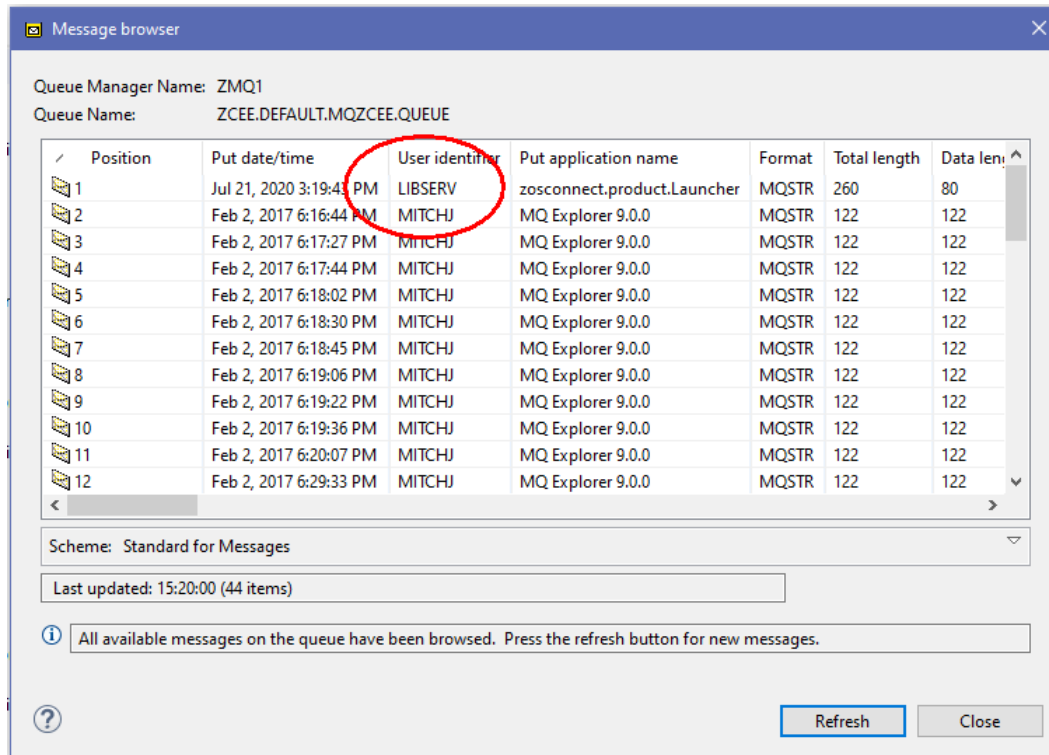
At the bottom, there are tabs for 'Definition' and 'Configuration', with 'Configuration' being the active tab.

These JNDI names were related to the actual queue manager and queue by the `jmsConnectionFactory` and `jmsQueue` configuration elements in the `server.xml` (see below).

```
<jmsConnectionFactory id="qmgrCf"
jndiName="jms/qmgrCf"
connectionManagerRef="ConMgr1">
  <properties.wmqJMS transportType="CLIENT"
    queueManager="ZMQ1"
    channel="LIBERTY.DEF.SVRCONN"
    hostName="wg31.washington.ibm.com"
    port="1422" />
</jmsConnectionFactory>

<jmsQueue id="q1" jndiName="jms/default">
  <properties.wmqJMS
    baseQueueName="ZCEE.DEFAULT.MQZCEE.QUEUE"
    CCSID="37" />
</jmsQueue>
```

- \_\_\_5. Open the MQ Explorer on the desktop and connect to queue manager *ZMQ1*. Expand the queues and select queue *ZCEE.DEFAULT.MQZCEE.QUEUE*. Right mouse button click and select *Browse Messages...* You should see the message just added in position 1 and the *User identifier* should be the identity under which the zCEE server is executing. The identity is not being propagated.



- \_\_\_6. Edit the include file configuration file *mqClient.xml* for the *myServer* server, e.g. */var/zcee/myServer/includes/mqClient.xml* and change the *useCallerPrincipal* property from *false* to *true*. see below:

```
<property name="useCallerPrincipal" value="true"/>
```

- \_\_\_7. Refresh the server's configuration by entering MVS command

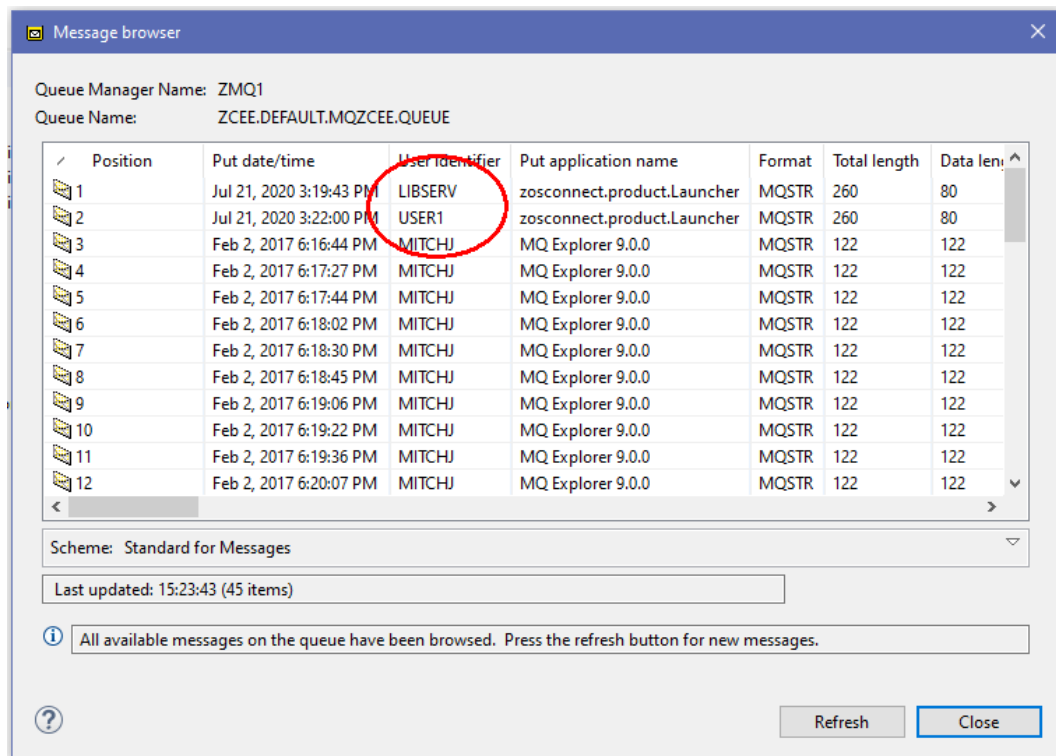
```
F BAQSTRT,REFRESH,CONFIG
```

- \_\_\_8. Enter the cURL command below:

```
curl -X POST -w '%{http_code}' --cacert certauth.pem --cert user1.p12:secret --cert-type P12 -
-header "Content-Type: application/json" --data @MQMessage.json
https://wg31.washington.ibm.com:9443/mqapi/queue
```

```
c:\z\admin>curl -X POST -w '%{http_code}' --cacert certauth.pem --cert
user1.p12:secret --cert-type P12 --header "Content-Type: application/json" --data
@MQMessage.json https://wg31.washington.ibm.com:9443/mqapi/queue
'204'
```

9. In the *MQ Explorer* session, use the **Refresh** button to redisplay the messages in queue *ZCEE.DEFAULT.MQZCEE.QUEUE*. You will see a new message on queue with the identity pass from the client. In this case the identity was based on the client certificate (*user1.pem*) used in the curl command.



Message browser

Queue Manager Name: ZMQ1

Queue Name: ZCEE.DEFAULT.MQZCEE.QUEUE

Position	Put date/time	User Identifier	Put application name	Format	Total length	Data length
1	Jul 21, 2020 3:19:43 PM	LIBSERV	zosconnect.product.Launcher	MQSTR	260	80
2	Jul 21, 2020 3:22:00 PM	USER1	zosconnect.product.Launcher	MQSTR	260	80
3	Feb 2, 2017 6:16:44 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
4	Feb 2, 2017 6:17:27 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
5	Feb 2, 2017 6:17:44 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
6	Feb 2, 2017 6:18:02 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
7	Feb 2, 2017 6:18:30 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
8	Feb 2, 2017 6:18:45 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
9	Feb 2, 2017 6:19:06 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
10	Feb 2, 2017 6:19:22 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
11	Feb 2, 2017 6:19:36 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122
12	Feb 2, 2017 6:20:07 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122

Scheme: Standard for Messages

Last updated: 15:23:43 (45 items)

All available messages on the queue have been browsed. Press the refresh button for new messages.

Refresh Close

## Summary

In this section, a simple REST client has been used to invoke an API which puts a message on an MQ queue. Initially the messages were written to the queue using the identity under the z/OS Connect server is running. A change to the *useCallerPrincipal* property for the underlying service asserts the identity used to authenticate to the z/OS Connect server on to the queue manager for MQ security checks.



## ***Enabling TLS security to a MQ queue Manager from a zCEE Server***

Adding TLS support to a MQ queue manager requires the creation of a key ring belonging to the identity under which the channel initiator address space is executing (look for message IEF695I in the channel initiator task JES messages). This key ring contains the personal and all the certificate authority certificates that will be used during TLS handshakes. The creation of the key ring and the connection of certificates to the key ring are done using the RACDCERT RACF command. Enabling TLS support in the channel initiator region itself will be done by changing a queue manger properties. This will require a restart of the channel initiator task

This exercise will begin going through the steps to enable TLS communications between a MQ queue manger and the z/OS Connect server.

### ***Creating MQ SAF resources***

In this scenario, the certificate authority used by organization has provided two certificates in advance. Both certificates currently reside in MVS data sets. One certificate is a personal certificate that will be used by CICS. This certificate was exported with a private key and will be used as a server certificate when CICS is acting a server endpoint or as a client or personal certificate when CICS is the client endpoint. The other certificate is the public certificate authority certificate that was used the sign the certificate assigned to CICS (this certificate only contains the public key).

- \_\_\_1. Browse data set *USER1.ZCEE30.CNTL*.
  
- \_\_\_2. Next, browse member **MQTLS**. You should see the RACF commands below. Submit the job for execution.

```

/* Create a CA certificate for MQ */
racdcert certauth gencert subjectsdn(cn('MQ CA') ou('ATS') +
o('IBM')) withlabel('MQ CA') keyusage(certsign) +
notafter(date(2022/12/31))

/* Create a server certificate for MQ TLS request */
racdcert id(MQSTC) gencert subjectsdn(cn('wg31.washington.ibm.com') +
ou('ATS') o('IBM')) withlabel('MQ CHIN') signwith(certauth +
label('MQ CA')) notafter(date(2021/12/31))

setropts raclist(digtring,digtnmap) refresh

/* Create a personal certificate for MQ TLS request */
racdcert id(user1) gencert subjectsdn(cn('user1') +
ou('ATS') o('IBM')) withlabel('MQ Client Cert') signwith(certauth +
label('MQ CA')) notafter(date(2022/12/31))

/* Create MQ key ring and connect CA and personal certificates */
racdcert id(MQSTC) addring(MQ.KEYRING)

racdcert id(MQSTC) connect(ring(MQ.KEYRING) +
label('MQ CA') certauth usage(certauth))

racdcert id(MQSTC) connect(ring(MQ.KEYRING) +
label('zCEE CA') certauth usage(certauth))

racdcert id(libserv) connect(ring(zCEE.KeyRing) +
label('MQ CA') certauth usage(certauth))

/* Connect default personal certificate */
racdcert id(MQSTC) connect(ring(MQ.KEYRING) +
label('MQ CHIN') default

racdcert certauth EXPORT(LABEL('MQ CA')) -
DSN('USER1.MQCACERT.PEM')

racdcert id(user1) export(label('MQ Client Cert')) -
DSN('USER1.MQCERT.P12') FORMAT(PKCS12DER) PASSWORD('secret')

setropts raclist(digtring,digtnmap) refresh

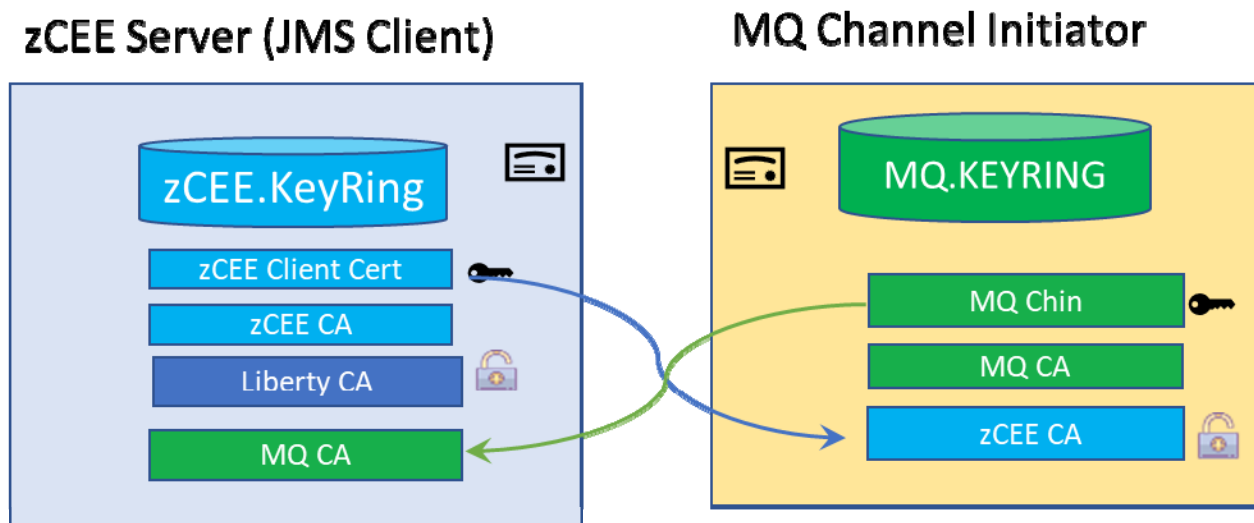
```

**Tech-Tip:** In summary, these commands create a new MQ certificate authority certificate and creates a server and personal certificate for MQ signed by this certificate.

A key ring is created for the MQ tasks and require personal and certificate authority certificates are connected to this key ring.

Finally, the MQ CA certificate and the personal certificate signed by the MQ CA certificate are exported to an MVS data set.

Below is visual representation of the MQ key ring just created.



### *Enabling MQ TLS support for a queue manager*

The next step is to update the queue manager attributes to enable support for TLS.

- \_\_\_1. Browse member *MQUTIL* in data set *USER1.ZCEE30.CNTL*

```
ALTER QMGR SSLKEYR(MQ.KEYRING) SSLTASKS(5) CERTLABL('MQ CHIN')
DEFINE CHANNEL ('LIBERTY.SSL.SVRCONN') REPLACE +
    CHLTYPE(SVRCONN) TRPTYPE(TCP) +
    SSLCIPH('TLS_RSA_WITH_AES_256_CBC_SHA256')
SET CHLAUTH('LIBERTY.SSL.SVRCONN') ACTION(REPLACE) +
    TYPE(SSLPEERMAP) ADDRESS('*') CHCKCLNT(ASQMGR) +
    SSLCERTI('CN=MQ CA,OU=ATS,O=IBM') SSLPEER('OU=ATS') USERSRC(CHANNEL)
```

- \_\_\_2. Submit the job for execution.
- \_\_\_3. Shutdown the channel initiator with MVS command **ZMQ1 STOP CHINIT**
- \_\_\_4. When task ZMQ1CHIN is no longer active (you may need to disconnect from the MQ Explorer), restart the channel initiator with MVS command **ZMQ1 START CHINIT**

- \_\_\_5. Edit the *server.xml* configuration file for the *myServer* server, e.g. */var/zosconnect/servers/myServer/server.xml* and change the include for *mqClient.xml* to be an include for file *mqClientTls.xml* see below:

```
<include location="{server.config.dir}/includes/mqClientTLS.xml" />
```

```
<include location="{server.config.dir}/includes/safSecurity.xml" />
<include location="{server.config.dir}/includes/ipicIDProp.xml" />
<include location="{server.config.dir}/includes/keyringMutual.xml" />
<include location="{server.config.dir}/includes/groupAccess.xml" />
<include location="{server.config.dir}/includes/shared.xml" />
<include location="{server.config.dir}/includes/mqClientTls.xml" />
```

The difference between *mqClientTls.xml* and *mqClient.xml* is the name of the channel to be used for communications was changed from *LIBERTY.DEF.SVRCONN* to *LIBERTY.SSL.SVRCONN* (see job

```
<jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
  connectionManagerRef="ConMgr1">
  <properties.wmqJMS transportType="CLIENT"
    queueManager="ZMQ1"
    channel="LIBERTY.SSL.SVRCONN"
    hostName="wg31.washington.ibm.com"
    sslcipherSuite="SSL_RSA_WITH_AES_256_CBC_SHA256"
    port="1422" />
</jmsConnectionFactory>
```

MQUTIL) and by the addition of a *sslcipherSuite* attribute.

- \_\_\_6. Enter MVS commands **F BAQSTR,REFRESH,CONFIG** to refresh the zCEE server's runtime configuration.
- \_\_\_7. Enter MVS commands **F BAQSTR,REFRESH,KEYSTORE** to refresh the zCEE server's runtime configuration.
- \_\_\_8. Enter the cURL command below:

```
curl -X POST -w '%{http_code}' --cacert certauth.pem --cert user1.p12:secret --cert-type P12 --header
"Content-Type: application/json" --data @MQMessage.json
https://wg31.washington.ibm.com:9443/mqapi/queue
```

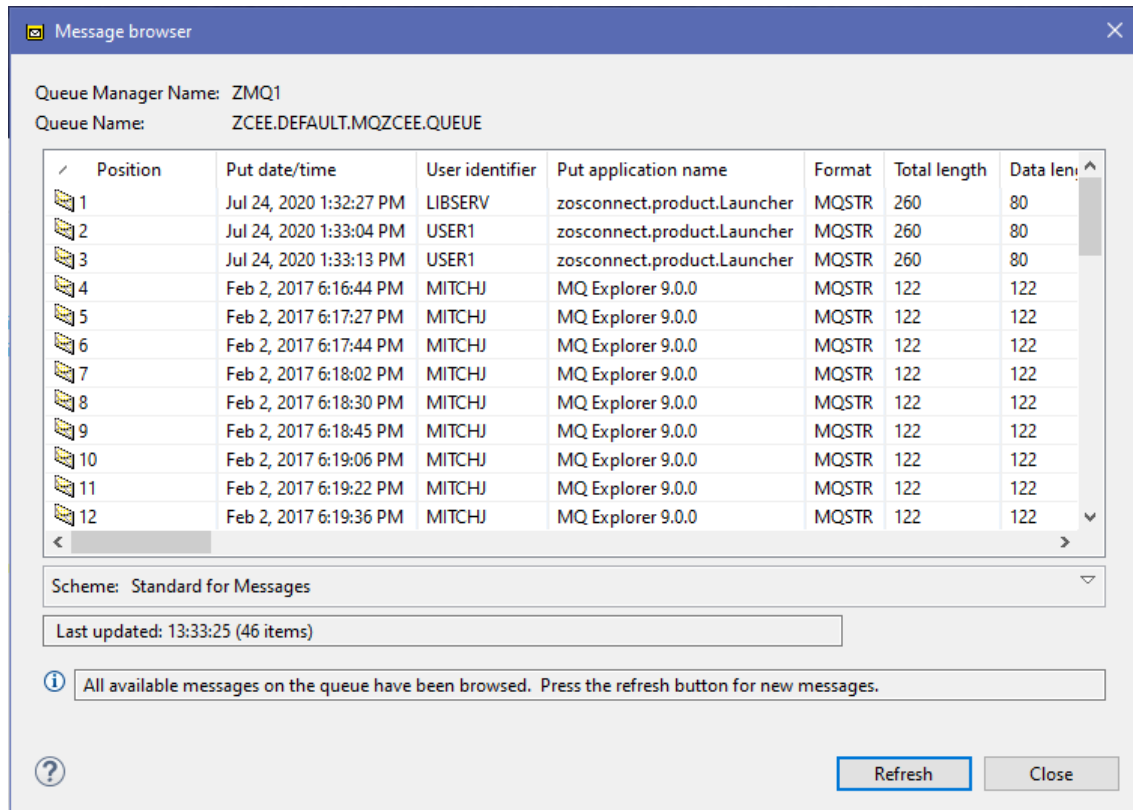
```
c:\z\admin>curl -X POST -w '%{http_code}' --cacert certauth.pem --cert
user1.p12:secret --cert-type P12 --header "Content-Type: application/json" --data
@MQMessage.json https://wg31.washington.ibm.com:9443/mqapi/queue
'204'
```

- \_\_\_9. Use MQ Explorer to display the channel status

Channels						
Filter: Standard for Channels						
Channel name	Channel type	QSG disposition	Overall channel status	Conn name	Transmission queue	MCA user ID
LIBERTY.DEF.SVRCONN	Server-connection	Queue manager	Running			
LIBERTY.SSL.SVRCONN	Server-connection	Queue manager	Running			
WAS.JMS.SVRCONN	Server-connection	Queue manager	Inactive			

of 15

- \_\_\_10. Browse the messages on the queue to see the presence of a new message a value for *User identifier* that corresponds to the client certificate used in the cURL command (user1.p12).



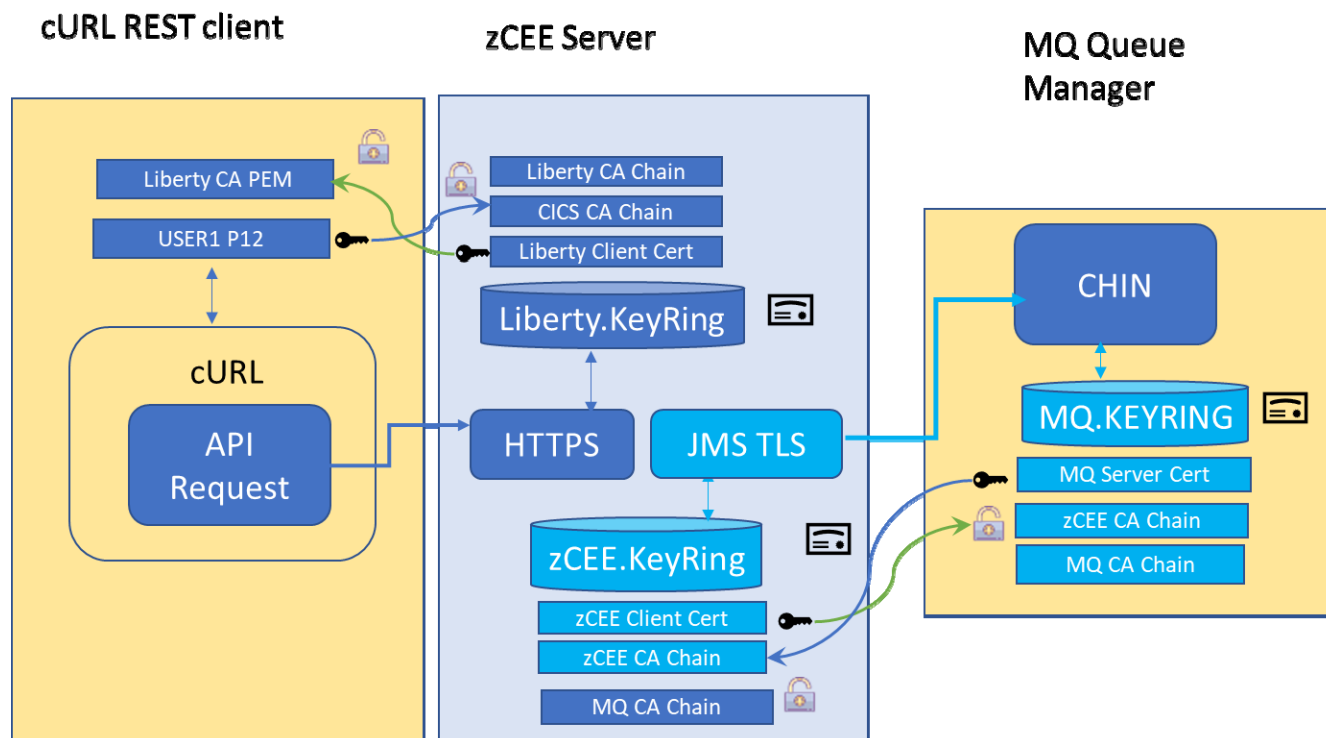
- \_\_\_11. Enter MVS command **ZMQ1 DISPLAY CHSTATUS(LIBERTY\*) MCAUSER SSLCERTU**

This will display the output below the value of local SAF identity associated with the remote client certificate used during the TLS handshake (*SSLCERTU*). The asterisk value for *MCAUSER* indicates the MCA user identity varies across these conversations.

```
CSQM201I ZMQ1 CSQMDRTC  DISPLAY CHSTATUS DETAILS
CHSTATUS (LIBERTY.SSL.SVRCONN)
CHLDISP (PRIVATE)
CONNAME (192.168.17.201)
CURRENT
CHLTYPE (SVRCONN)
STATUS (RUNNING)
SUBSTATE (RECEIVE)
STOPREQ (NO)
RAPPLTAG (zosconnect.product.Launcher)
SSLCERTU (LIBSERV)
MCAUSER (*)
END CHSTATUS DETAILS
CSQ9022I ZMQ1 CSQMDRTC ' DISPLAY CHSTATUS' NORMAL COMPLETION
```

*The results raise a question. Why is the SSLCERTU identity LIBSERV? For an explanation see below.*

The diagram below shows the identity that flows the cURL RESR client. Since the connection between cURL and the zCEE server is protected by TLS, the identity associated with the client certificate (USER1.P12) can be asserted from CICS to the zCEE server for an authorization check for executing the API. This identity does not flow to the API provider. Since it is a TLS connection and mutual authentication is enabled between the zCEE server, the authenticated identity is the RACF identity mapped to the zCEE server client certificate (LIBSERV). This identity is used authorization checks and, in this case, propagated to the MQ queue manager. Since the useCallerPrincipal attribute is set to true the zCEE authentication identity of USER1 is sent in the MCAUSER field

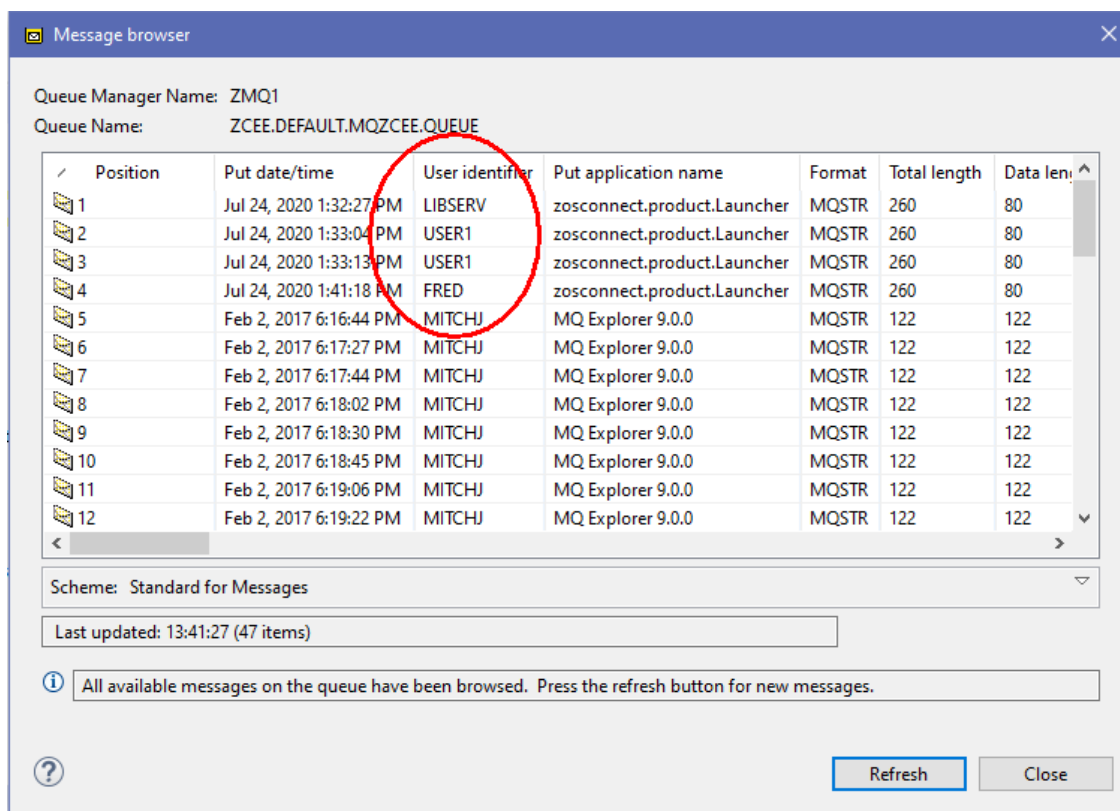


12. Enter the cURL command below:

```
curl -X POST -w '%{http_code}' --cacert certauth.pem --cert fred.p12:secret --cert-type P12 --header "Content-Type: application/json" --data @MQMessage.json https://wg31.washington.ibm.com
```

```
c:\z\admin>curl -X POST -w '%{http_code}' --cacert certauth.pem --cert fred.p12:secret --cert-type P12 --header "Content-Type: application/json" --data @MQMessage.json https://wg31.washington.ibm.com:9443/mqapi/queue
'204'
```

- \_\_\_13. Browse the messages on the queue to see the presence of a new message a value for *User identifier* that corresponds to the client certificate used in the cURL command (fred.p12).



## Summary

In this section the use of TLS was enabled between a z/OS Connect server and an MQ queue manager by adding the required RACF resources and updating the queue manager attributes and the JMS connection factory using in the server's configuration file.