## Lab 2 - Protecting message data with MQ Advanced Message Security

This lab introduces the student to the components of IBM MQ Advanced Message Security (MQ AMS). Students will learn about:

* Protection policies
* Levels of protection
* MQ AMS interceptors
* Administrative tools
* Architecture and deployment

The student will exercise MQ AMS message protection in numerous scenarios including:

* Protecting messages on a local queue
* Protecting messages on a remote queue
* Protecting messages on a cluster queue
* Removing rogue messages from the MQ network

## Key Concepts

MQ AMS augments IBM MQ security by providing data protection of messages. Three “Qualities of Protection” (QOP) are provided:

* Integrity (signed)

IBM MQ messages are signed when put on a queue and the signature embedded into the message is verified when the message is retrieved. The signature identifies the sender of the message and ensures that the content of the message has not been changed.

MQ AMS uses PKCS #7 data-envelope to envelop a message in a digital signature.

* Privacy (sealed)

IBM MQ messages are signed as above but the content of the message is also encrypted. The key used to encrypt messages is itself encrypted for each potential recipient so that the messages can be retrieved by one or more recipients.

MQ AMS uses PKCS #7 data-envelope to envelop a message in a digital signature.

* Confidentiality (encrypted only)

IBM MQ messages are encrypted end-to-end using a symmetric key and are not digitally signed. This reduces the amount of asymmetric key operations by half and improves performance.

Optionally the encryption keys can be reused for multiple messages going to the same recipient. The second and subsequent messages incur no asymmetric key operations which keeps performance similar to plaintext/unprotected message throughput.

Data protection is provided by various interceptors relying on configuration files, PKI keystores, and certificates. This is in addition to the standard protection available for IBM MQ objects using the IBM MQ Object Authority Manager (OAM) or SAF (via a product such as IBM’s RACF) on z/OS. The diagram below depicts the overall architecture:

## 

## Interceptors

MQ AMS implements security by intercepting IBM MQ API calls. The following APIs are supported:

* Message Queue Interface (MQI)
* Java Message Service (JMS)
* IBM MQ Java Service (JMS)
* IBM MQ Java Classes

Three interceptors are provided:

* IBM MQ server interceptor

To protect messages put or get by applications using the mqm library and using the interprocess communication (IPC), also called bindings mode.

This interceptor will also protect messages for Java applications (either using the MQ Java API or JMS) connecting in MQ bindings mode.

* IBM MQ C client interceptor

To protect messages put or get by applications using the mqm client library and therefore connecting using a (non-Java) MQ client connection.

* IBM MQ Java interceptor (JMS and MQ Java)

To protect messages put or get by Java applications (MQ Java API or JMS) using either client connections or bindings mode.

Note that if you use this interceptor for applications using Java bindings mode, you should consider disabling the server interceptor to avoid signing/encrypting messages twice.

### Public Key Infrastructure (PKI) and Certificates

MQ AMS uses Public Key Infrastructure (PKI) identities to represent users or applications. The identities are used to sign and encrypt messages.

An identity is represented by the distinguished name (DN) of a personal X.509 (v2/v3) certificate associated with the signed and/or encrypted message. To authenticate this authority, the user or application must have access to a keystore containing a public certificate and an associated private key.

A mechanism is provided to map the OS identity of the user or application to the specific keystore file.

### Object Authority Manager (OAM)

MQ AMS provides message security. It does not provide authorization services for IBM MQ objects. The task to authorize access to IBM MQ resources (queue manager, queues and other objects) remains the responsibility of the IBM MQ Object Authority Manager (OAM) or RACF (or other SAF) on z/OS. When an interceptor intercepts the MQ API calls from an application, it delegates the authorization request to the IBM MQ authority service.

### Error Handling

MQ AMS defines an error handling queue (**SYSTEM.PROTECTION.ERROR.QUEUE**) to manage messages that contain errors or messages that cannot be decrypted. If a received message does not meet the security requirements for the queue it is on, then the message is moved to the error handling queue.

The following is a non-exclusive list of possible reasons why a message might be sent to the error handling queue:

* Quality of protection mismatch
* Decryption error
* PDMQ header error
* Size mismatch
* Encryption algorithm strength mismatch
* Unknown error

The system error handling queue can optionally be defined as an alias queue pointing to another queue.

### Supported Platforms

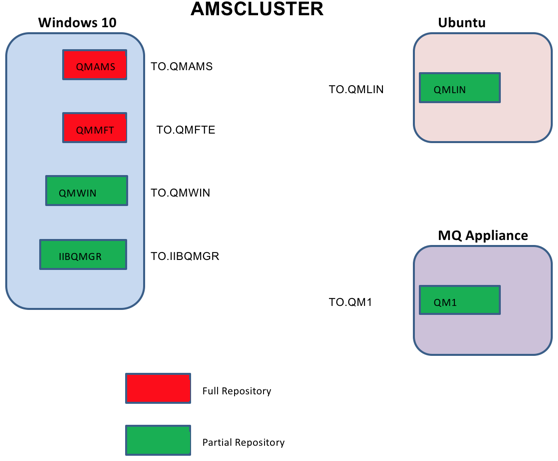
For the latest and complete list of supported platforms refer to the link below:

[MQ 9 Advanced Message Security supported platforms](http://www-01.ibm.com/support/docview.wss?uid=swg27047751)

[MQ 9 Knowledge Center – Advanced Message Security](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.sec.doc/q014580_.htm)

## Lab Environment

The lab environment for the PoT is shown below. The environment should still be running on Skytap.



QMAMS and IIBQMGR are the only queue managers used in this Lab 2. For Lab 2, the student should only be concerned with the environment shown above.

Table 1: Names used in this lab

|  |  |  |
| --- | --- | --- |
| Host name | IP | Purpose |
| win10 | 10.0.0.2 | Windows 10 |

|  |  |  |
| --- | --- | --- |
| Queue Manager Name | Host Address | Port |
| QMAMS | win10 | 1415 |
| IIBQMGR | win10 | 1424 |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Queue Name | Queue Manager | Queue Type |
| ORDERQ | QMAMS | LOCAL |
| REMOTE\_ORDERQ | IIBQMGR | REMOTE |
| CLUSTER\_ORDER | QMAMS, IIBQMGR | CLUSTER QUEUE |

Table 2: UserIDS used in this lab

|  |  |  |
| --- | --- | --- |
| User-ID | Group | Purpose |
| ibmdemo | Ibmdemo, mqm | Windows and MQ ibmdemo |
| msgsend | mqusers | IBM MQ application user-id |
| msgrecv | mqusers | IBM MQ application user-id |
| msgspy | mqusers | IBM MQ application user-id |

### Important Points

* To login to the Windows image use the *ibmdemo* user-ID
* The password for ibmdemo is *passw0rd* \* The password for all other users is *password*
* The password for all CMS key databases and JKS keystores is *password*
* User-IDs msgsend, msgrecv and msgspy each have a CMS key database and a JKS keystore already setup in their respective directories. (You created these in the previous lab.)

### MQ environment used in this lab

The IBM MQ environment compromises the following:

* IBM MQ v9.1 (all components installed)
* IBM MQ Explorer on the desktop (icon)
* Two queue managers:
* QMAMS (port 1415)
* IIBQMGR (port 1424)
* Cluster Sender and Cluster Receiver channels to connect the above queue managers in a Cluster \*
* Various other MQ objects to be used for the user scenarios

### Programs used in this lab

* To test various scenarios with MQ AMS, the following sample applications are available:
* amqsput (IBM MQ C sample – server)
* amqsputc (IBM MQ C sample – client)
* amqsget (IBM MQ C sample – server)
* amqsgetc (IBM MQ C sample – client)

These samples come with the IBM MQ product and are located in \*C:\\Program Files\\IBM\\MQ\tools\c\Samples\Bin.

## Installation and Configuration

IBM MQ AMS is included in the IBM MQ Advanced V9 package and is another option on the IBM MQ installer.

### Installation

MQ AMS has been pre-installed on the VMWare image. You can view the [MQ 9 Knowledge Center](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.ins.doc/q009140_.htm) for installation instructions, if interested.

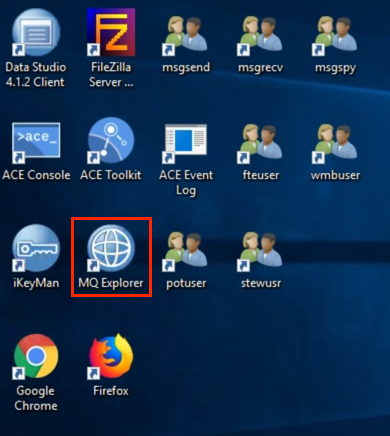
For this lab, we will begin with the configuration steps.

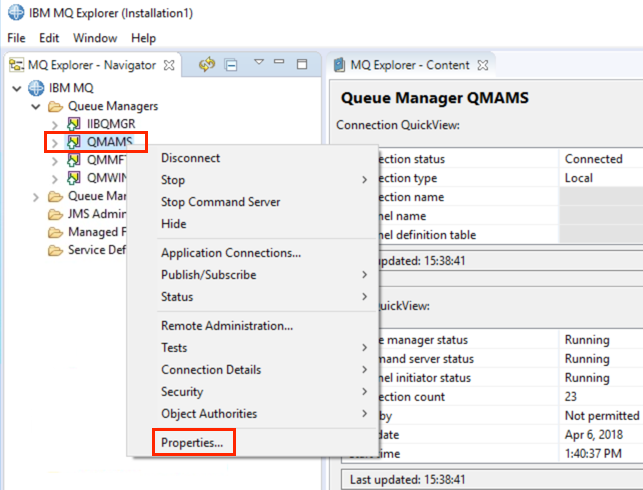
### Procedures

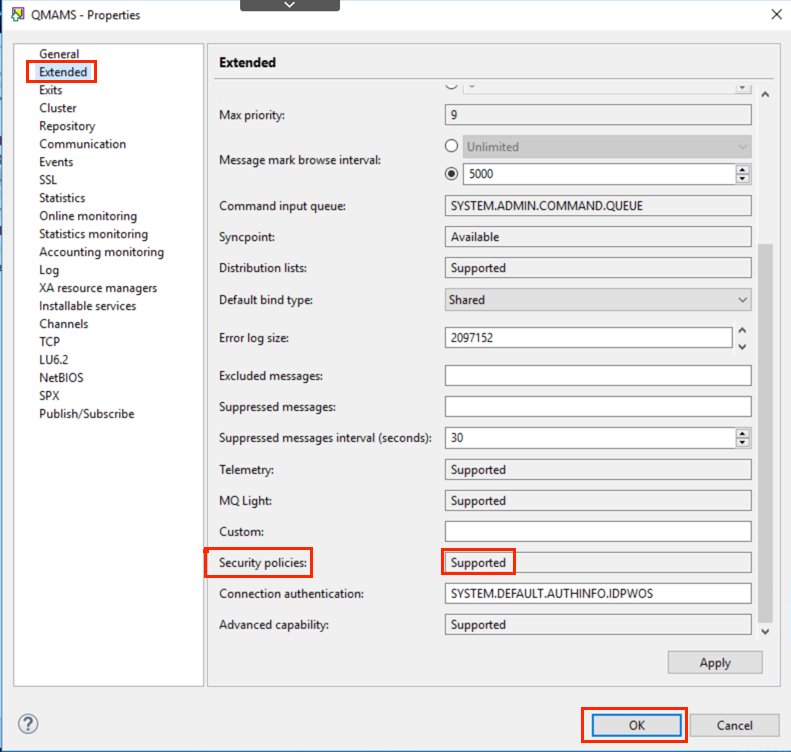
* Configuring MQ AMS requires that you do the following:
* Create IBM MQ queues required by MQ AMS
* Update IBM MQ configuration files (server interceptor)

You will now configure MQ AMS on the Windows VMware image by following the steps outlined below:

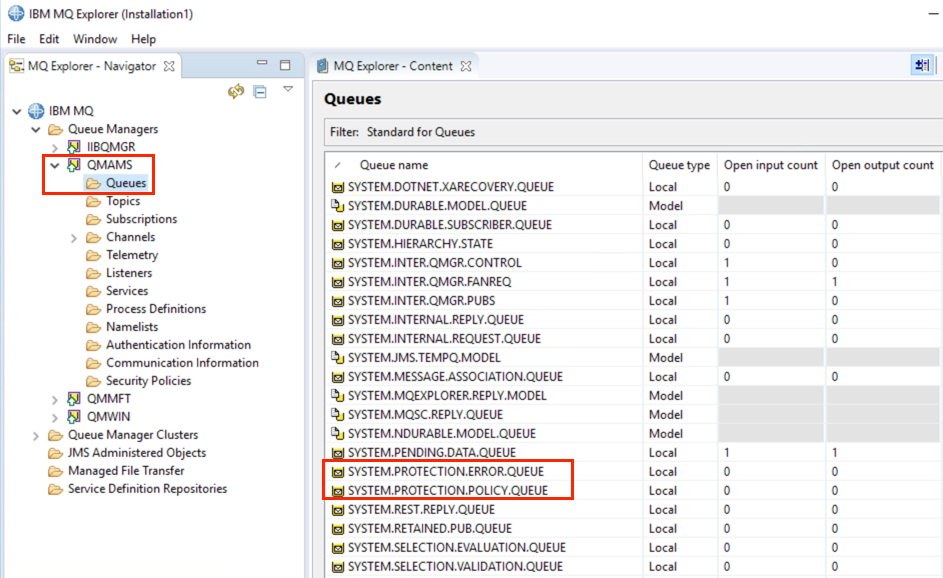
1. You should be logged on as **ibmdemo**.
2. Start MQ Explorer by double-clicking the icon on the desktop.



1. Expand **Queue Managers** and right click Queue Manager **QMAMS** and select **Properties**.  
   
2. Click **Extended**. Scroll to the bottom where you find that **Security Policies** attribute is set to supported.
3. Click **OK** to close the Properties window.



1. Expand queue manager **QMAMS**.
2. Click **Queues** under Queue Manager **QMAMS**.
3. Ensure that the “Show System Objects” toggle icon is selected, and scroll down until you find **SYSTEM.PROTECTION.\*.** queues.



1. When AMS is selected as part of the installation, the Security Policies “Supported” attribute is set and the following two queues are defined:

* SYSTEM.PROTECTION.POLICY.QUEUE

This queue stores security policy definitions.

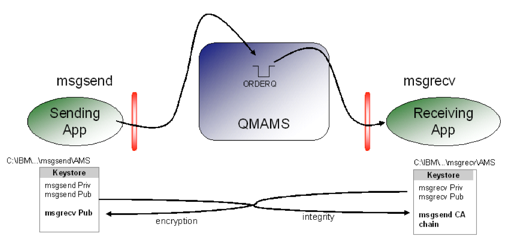
* SYSTEM.PROTECTION.ERROR.QUEUE

This queue is used as the error handling queue.

1. You should repeat steps 3 - 8 for queue manager **IIBQMGR**.

## Creating keystore.conf

When the interceptor is to encrypt the message, it requires the private key of the sending user. Thus, a mapping of the key database user identities to public and private keys needs to be created. In the real system, where users and applications are dispersed over several machines, each user would have their own private keystore. Similarly, in the previous lab, we created separate keystores and certificates for the users msgsend, msgrecv, and msgspy. For the scenario where the user msgsend sends an encrypted message to msgrecv, this appears as follows:

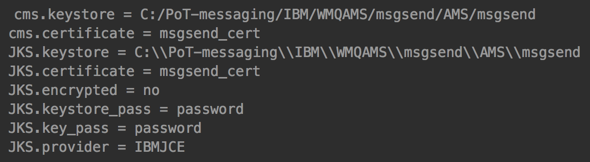


Once the key databases and certificates are created (Lab 1), you must point MQ AMS interceptors to the directory where they are located. This is done via the **keystore.conf file**, which holds that information in the plain text form. A separate *keystore.conf* file must be created for each user, so it is advisable to create it next to the key database. The ***keystore.conf*** files have been pre-configured for users **msgsend**, **msgrecv**, and **msgspy**. You will find them in **C:\PoT-messaging\<userid>\.mqs**.

The content of keystore.conf is of the form:



In this lab, the content of the keystore.conf for msgsend is as follows:



|  |  |
| --- | --- |
| sign-info | The Path to the keystore file must be provided with no file extension. The password is retrieved from the stash file in the .CMS format.  If you choose to use the Java keystore file formats (.JKS, .JCEKS), additional lines must be added to the keystore.conf. HOME/.mqs/keystore.conf or C:\Users\<user\_name>\.mqs\keystore.conf for Windows is the default location where MQ AMS searches for the keystore.conf file.  However, users can modify it by providing an environment variable MQS\_KEYSTORE\_CONF=<config\_file\_path> or by specifying a system property using the -DMQS\_KEYSTORE\_CONF=<config\_file\_path> command for Java. We are using the environment variables in this lab. |

### Procedures

As described above, to setup an application to run with MQ AMS, a configuration file (*keystore.conf*) must be created in sub-directory .mqs of the user home directory.

Alternatively, environment variable MQS\_KEYSTORE\_CONF may be defined to point to the location of **keystore.conf**. In this lab, the environment variable method is used. That environment variable has been set for users msgsend, msgrecv, and msgspy with the path **C:\<userid>\.mqs**.

Default locations of the keystore.conf file are:

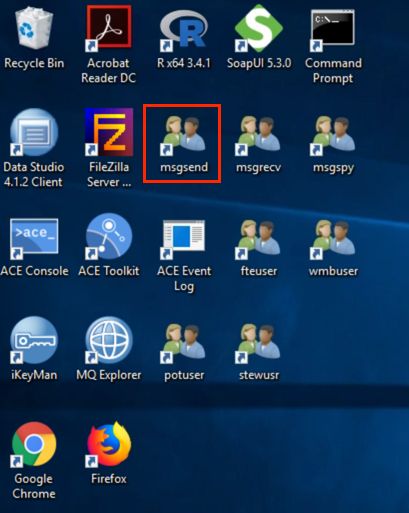
Windows C:<user\_name>.mqs.conf

Unix/Linux /home//.mqs/keystore.conf

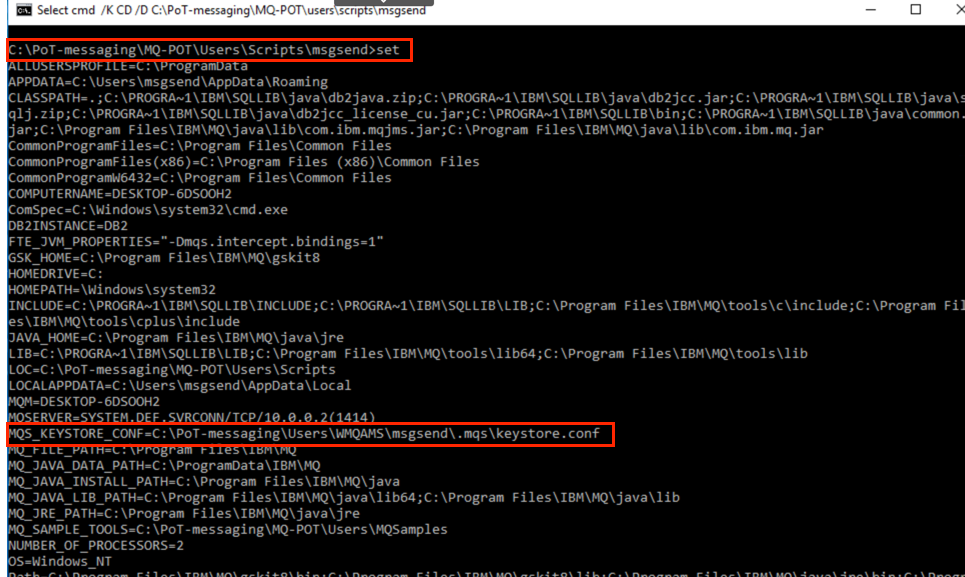
Configuration file keystore.conf is used by the interceptors to locate and access the user’s keystore(s).

Configuration files have already been setup for user-IDs msgsend, msgrecv, and msgspy.

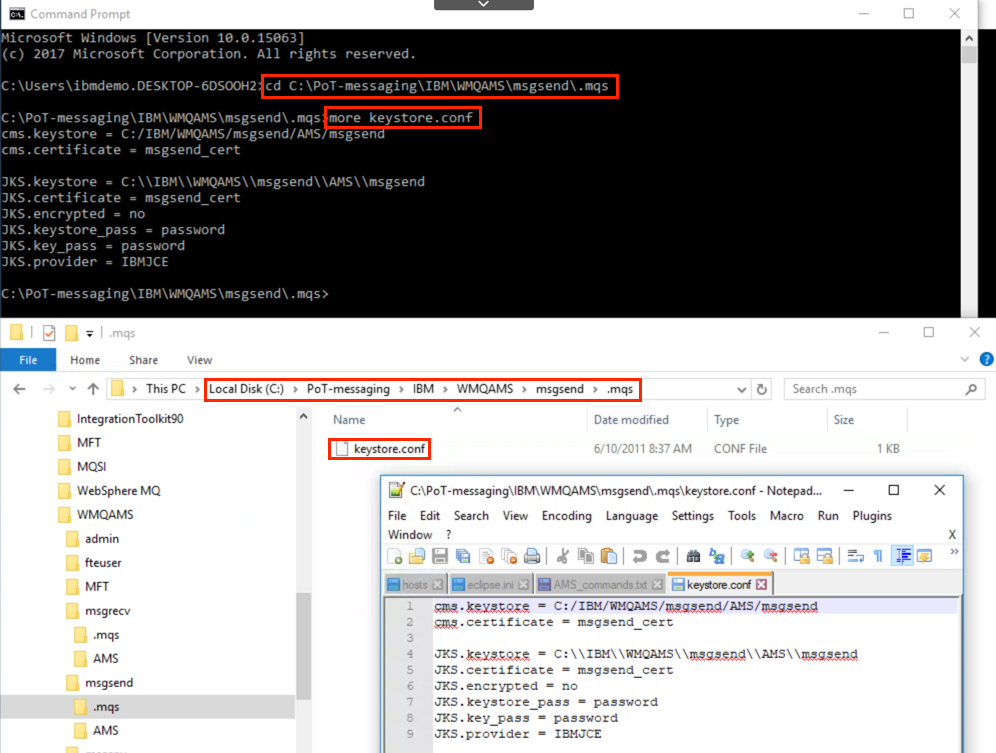
1. Use Windows Explorer to navigate to C:.mqs and double-click the keystore.conf file in that directory to view it.
2. Open a command prompt for msgsend by double-clicking its user icon on the desktop.



1. If you are prompted for a password, you will need to enter the “password” for msgsend.
2. Enter the *set* command to display msgsend’s environment variables.
3. Locate the environment variable MQS\_KEYSTORE\_CONF. Notice that it is not the default value mentioned earlier.



1. If not already open, open a command prompt and change to the directory C:-messaging.mqs. (Do not use the command prompt for msgsend. Open a command prompt as *ibmdemo* using the task bar icon.)
2. Type the following command in DOS command prompt to review the keystore.conf file for user-ID msgsend:

* more keystore.conf
* or view in Windows Explorer.
* 

1. You may repeat steps 3 – 6 for the other users msgrecv and msgspy to view their keystore.conf files. The configuration file points to two different keystores. The first one is a CMS keystore used by non-Java interceptors (server and compiled MQ client). The second keystore is a JKS and is used by the Java interceptors. Notice the forward slashes in the CMS.keystore path and double back slashes in the JKS.keystore path. These are standard ways to define paths for Java programs on Windows.

Notice that the JKS passwords are in the clear. Passwords in the configuration file may be encrypted by using class com.ibm.mq.ese.config.KeyStoreConfigProtector as described in the documentation. If you are interested in this option, see the URL:

<http://www-01.ibm.com/support/knowledgecenter/search/com.ibm.mq.ese.config.KeyStoreConfigProtector?scope=SSFKSJ_9.0.0>

|  |  |
| --- | --- |
| sign-info | JKS and JCEKS are also supported keystore database formats. |

Your system is now fully configured and interceptors for all server and Java environments are enabled. Note that by default MQ AMS will not sign or encrypt any messages until a policy is defined for a queue.

## Setting Protection Policies

With the queue manager created and interceptors prepared to intercept messages and access encryption keys, we can start defining protection policies for our queues. MQ AMS will not sign or encrypt any messages until an actual policy is defined for a queue. This means that, by default, message protection is not in force even though interceptors may be enabled. To define, modify or delete policies, you can use either command line tools or IBM MQ Explorer (on Windows and Linux) providing that you have installed the MQ AMS plugin. When defining a policy, the following information needs to be specified:

* The name of the policy which is the name of the queue your application(s) will put or get messages to/from. A policy name is the same as its protected queue name.
* The quality of protection required:
* NONE, if no protection is needed
* INTEGRITY, if messages on the queue should be signed
* PRIVACY, if messages on the queue should be signed and encrypted
* CONFIDENTIALITY, if messages on the queue should be encrypted without signing
* The signature algorithm to use to sign the messages if quality of protection INTEGRITY or PRIVACY is selected:
* NONE
* MD5
* SHA1
* SHA256
* SHA384
* SHA512
* The encryption algorithm used to encrypt messages if quality of protection PRIVACY or CONFIDENTIALITY is selected:
* NONE
* RC2
* DES
* Triple DES
* AES128
* AES256
* The recipient list, which is the list of certificate distinguished names (DN) of potential receivers of the messages.
* The signature DN checklist, which is the list of distinguished names (DN) to be validated during message retrieval. This is the DN of the sender.
* Toleration which specifies whether a policy can be ignored or must be enforced. Toleration is optional and facilitates staged roll-out, where policies were applied to queues but those queues may already contain messages that have no policy, or still receive messages from remote systems that do not have a security policy.

Policies are stored in IBM MQ queue **SYSTEM.PROTECTION.POLICY.QUEUE**.

An application’s userID needs to have browse authority on the above queue so that the interceptor is able to query the queue and retrieve any policy defined for the queue the application is trying to access.

### Administration using command line tools

This section is included as documentation. There are no exercises to be performed here. Do not issue the commands here. They will be exercised in section.

Two command line tools are provided to display and to create, modify or delete policies. These commands have names and syntax based upon standard MQ line commands. They are located in \*\*C:Files\*:

* dspmqspl

To display and dump policies  
Usage:

* dspmqspl -m \<QMGR> \[-p \<Policy Name>] \[-export]
* {% include note.html content=“The -export option will generate the policies in the setmqspl format.” %}
* setmqspl

To define, change and delete policies Usage:

* setmqspl -m \<QMGR> -p \<Policy Name> (-remove | -s \<signing algorithm> [-a \<signer DN>]\\* [-e \<encryption algorithm> [-r \<receiver DN>]+][-t <0|1>])The parameters surrounded by <> are mandatory, while those surrounded by [] are optional. Each policy name must be the same as the queue name it is to be applied to. The following is an example of policy defined on the ORDERQ queue, signed by the user msgsend using the SHA1 algorithm, and encrypted using the 256-bit AES algorithm for the user msgrecv: setmqspl -m QMAMS -p ORDERQ -s SHA1 -a "CN=msgsend,O=ibm,C=US" -e AES256 -r "CN=msgrecv,O=ibm,C=US" {% include warning.html content=“The DNs must match exactly those specified in the receptive user’s certificate from the key database.” %}

### Administration using MQ Explorer IBM MQ Explorer provides a new option under Queue Managers > Security Policies. This can be used to display or modify Security Policies for AMS on all platforms. Clicking Security Policies will display the list of policies currently defined.

1. Bring the MQ Explorer into focus. 2. Locate Security Policies in the object list under QMAMS. As you can see, there are no policies defined.

* To define a new policy, right-click Protection Policies and select New, Message Protection Policy.
* To modify an existing policy, right click the existing policy and select Properties.
* To delete a policy, right-click the policy and select Delete.

## User scenarios

### Signed messages on local queue (No signature DN checklist)

In this first scenario, the userID **msgsend** puts a message on queue **ORDERQ** and the userID **msgrecv** receives that message. A policy is defined to use quality of protection **Integrity** (messages are signed) but no signature DN checklist is provided. Then **msgsend** puts its own message on queue **ORDERQ** and **msgrecv** receives the message as well. This scenario is quite simple, and probably not representative of most “real world” environments, because both userIDs are on the same physical machine, and using the same MQ queue manager. However, it introduces you to the basic commands and architecture.

1. Sign-on each user.

* There are a number of shortcuts on the desktop for logging in as a specific user and opening a command prompt. Double-click each of the three shortcuts for msgsend, msgrecv, and msgspy. The first time, you may need to enter the password (password) for each user.  
  1. Arrange the windows as shown in the screen capture below:
  2. The shortcuts you have just clicked for each user also sets up the environment for use of MQ and MQ AMS and their personal keystores. Command files have been created for each user. The commands are in the directory \*\*C:-messaging-POT\*. The commands follow the pattern:
* *setenv-<user\_name>* example: setenv-msgsend
  1. The appropriate environment command for each user has been defined on the shortcut and was executed when you clicked the shortcut. Enter the set command at the command prompt to show the environment variable settings for each user.

1. Authorize users to MQ objects.

* As mentioned earlier, MQ AMS is fully complementary to IBM MQ and it delegates access authorization to the IBM MQ Object Authority Manager. This means AMS is only protecting the message contents, and it assumes the MQ ibmdemo has taken the necessary steps to protect access to the queue manager itself and the QM objects such as queues, channels, etc. Therefore, the first step is to MQ authorize the user-IDs to connect to the queue manager and be able to open queue ORDERQ for PUT/GET. a. Return to the Command-line prompt for **ibmdemo**. from the desktop and enter the following commmands.
* {% include note.html content=“Copy using the button on the commands and paste them into command line instead of typing them.” %}
* The following standard MQ ibmdemo command allows access to the queue manager for a few selected userIDs: setmqaut -m QMAMS -t qmgr -p msgsend -p msgrecv -p msgspy +connect +inq The next command gives specific MQGET and MQPUT authorization to the queue that you’ll be using: setmqaut -m QMAMS -t q -n ORDERQ -p msgsend -p msgrecv –p msgspy +get +put
* {% include note.html content=“You’ll note above that you are giving the msgsend and msgrecv userIDs as well as the msgspy userID access to MQ and ORDERQ. Clearly in a protected environment, you would not give a non-trusted ID such as “msgspy” any access to an MQ object, but we are providing the MQ access here so that you’ll be able to test the AMS level of access later on." %}
  1. Additionally, an application’s user-ID via the interceptor will need to gain access to queue **SYSTEM.PROTECTION.POLICY.QUEUE** to retrieve the policy information and to queue **SYSTEM.PROTECTION.ERROR.QUEUE** which provides error handling. Still using the **ibmdemo** userID, issue the following commands: ``` setmqaut -m QMAMS -t q -n SYSTEM.PROTECTION.POLICY.QUEUE -g mqusers +browse
* setmqaut -m QMAMS -t q -n SYSTEM.PROTECTION.ERROR.QUEUE -g mqusers +put
* {% include note.html content=“Note that in the above command, you are giving access to the Windows user group “mqusers” which includes the three userIDs msgsend, msgrecv and msgspy." %}

1. Define an MQ AMS policy.

* The commands above established MQ authorization to the MQ objects. You’ll now define additional AMS protection above and beyond the MQ protection.
  1. Define the policy for queue ORDERQ. To define a policy, you need to use a user-ID which is part of the mqm group. Use userID ibmdemo to define a policy for queue ORDERQ using the SHA1 signing algorithm.
* setmqspl -m QMAMS -p ORDERQ -s SHA1
  1. Using the **ibmdemo** userID, display the policy you just created.
* dspmqpl -m QMAMS -p ORDERQ

1. Using the userID **msgsend** (in other words, from the Windows command prompt window opened with this userID), verify that the *MQS\_KEYSTORE\_CONF* environment variable for msgsend’s keystore is correctly set by typing the following command:

* set
* It should have been set by the setenv-msgsend command when logging on. If the variable is not set, set it with this command:  
  set MQS\_KEYSTORE\_CONF=C:\PoT-messaging\IBM\WMQAMS\msgsend\.mqs\keystore.conf
  1. Still using userID msgsend’s command prompt, use the sample C program amqsput as follows to put a message on the ORDERQ queue:
* amqsput ORDERQ QMAMS
* Enter a test message (example shown below) and hit Enter.
* This is a test message for AMS policy on ORDERQ

1. Verify that the message put to the queue has actually been signed. An alias queue has been defined for ORDERQ called **ORDERQ\_AL**. We did not give any authorities to users to access the alias queue, but it can be accessed by the mqm group. Since the MQ Explorer is running under the logged in userID of ibmdemo and ibmdemo is in the mqm group, we will be able to browse the ORDERQ queue by using the ORDERQ\_AL queue.
   1. In MQ Explorer, right-click the queue ORDERQ\_AL and select Browse Message.
   2. Double-click the message and select Data to view the message and verify that it is, in fact, signed. The beginning of the message data should start with the PDMQ header and embedded in the PDMQ header which contains the signature you should see the message (unencrypted) put on the queue by user msgsend (‘This is a test message’).

* Click Close to dismiss the display window. We can see more of the message if we use RFHutil to view the message.
  1. Start RFHutil by double-clicking the the shortcut on the desktop.
  2. Select the QMAMS queue manager and ORDERQ\_AL queue from the drop-down menus, and then click the Browse Q button.
* (Note the “Msg browse from ORDERQ\_AL length=xxx” at the bottom of the window).
  1. Then click the Data tab at the top of the window to view the message and verify that it is, in fact, signed. The beginning of the message data should start with the PDMQ header and embedded in the PDMQ header which contains the signature you should see the message put on the queue by user msgsend (‘This is a test message’). Notice that the message is signed by observing the “garbage” characters.

1. Running amqsget under user-ID msgrecv should retrieve the message. The integrity of the message will be verified.
   1. Start up the userID msgrecv user session using the shortcut on the desktop for msgrecv. (There should be an open command prompt for msgrecv). This sets the environment variable for msgrecv’s keystore with the setenv-msgrecv command.
   2. Still using userID msgrecv command prompt and sample C program amqsget, get a message from the ORDERQ queue.

* amqsget ORDERQ QMAMS
* {% include note.html content=“amqsget will exit automatically after 30 seconds, if no messages are available.” %}

1. Now repeat steps 3 thru 6 with the userIDs reversed. In other words try to put the message using msgrecv and get the message using msgsend. You will notice that user msgrecv can put to the ORDERQ queue and user msgsend can still retrieve the message. Does this surprise you? Think about the protections that you’ve defined thus far.
2. The reason the reverse (“wrong way round”) worked is:
   1. The security policy ORDERQ specifies “Accept signed messages from any originator”. This means that all messages must be signed, but that no selection/rejection is made based on the originator.
   2. The security policy ORDERQ does not sign the message, meaning that any recipient can accept the message.
3. Let’s assume a message is put to queue ORDERQ but arrives from another queue manager IIBQMGR. You can use the remote queue REMOTE\_ORDERQ on IIBQMGR to send the message to queue ORDERQ on QMAMS. This is illustrated below:

* The server interceptor is not enabled on queue manager IIBQMGR so the message put on queue ORDERQ on QMAMS is not signed. When user msgrecv attempts to get the message from queue ORDERQ it will fail because the message is not signed. The message will be sent to the MQ AMS error queue (SYSTEM.PROTECTION.ERROR.QUEUE).
* In order for this to work, you will now also need to set a policy on the remote queue REMOTE\_ORDERQ.
* This is because remote queues are only fully protected when setting the same policy on the remote queue and on the local queue to which it points. Here is the explanation from the Information Center:
* When a message is put into a remote queue, MQ AMS intercepts the operation and processes the message according to a policy set for the remote queue. For example, for an encryption policy, the message is encrypted before it is passed to MQ to handle it. After MQ AMS processes the message put into a remote queue, MQ puts it into associated transmission queue and forwards it to the target queue manager and target queue.
* When a GET operation is performed on the local queue, MQ AMS tries to decode the message according to a policy set on the local queue. In order for the GET operation to succeed, the policy used to decrypt the message must be identical to the one used to encrypt it. Any discrepancy will cause the message to be rejected.

1. As **ibmdemo**, run the following command to set the MQ AMS policy on the remote queue to match that on ORDERQ:

* setmqspl -m IIBQMGR QMAMS -p REMOTE\_ORDERQ -s SHA1

1. Run the following commands to authorize and then put/get some messages:
   1. From userID **ibmdemo**, authorize the msgspy userID for MQ access to the IIBQMGR.

* setmqaut -m IIBQMGR -t qmgr -p msgspy +connect +inq
* setmqaut -m IIBQMGR -t q -n REMOTE\_ORDERQ -p msgspy +put
  1. From userID msgspy, put message on REMOTE\_ORDERQ:
* amqsput REMOTE\_ORDERQ IIBQMGR
* This is a test message from msgspy on IIBQMGR to remote queue ORDERQ on QMAMS
  1. Now from userID msgrecv, try to receive the message from this AMS protected queue.
* amqsget ORDERQ QMAMS
* What does the 2063 reason code indicate? You might want to type “mqrc 2063” to get a quick interpretation of the error code. For further details, you could search in the InfoCenter.
* Now use the MQ Explorer to check the status of this queue. How many messages are in the ORDERQ now?
  1. Browse the **SYSTEM.PROTECTION.ERROR.QUEUE** queue to see the message.
* From userID **ibmdemo**:
* amqsbcg SYSTEM.PROTECTION.ERROR.QUEUE QMAMS
  1. Using Windows Explorer, navigate to the following directory. **C:\***
* MQ AMS errors are written to the queue manager error logs. Locate the most current error file – **AMQERR01**. Scroll to the bottom to find the error.
* MQ AMS intercepted the message on the MQGET and did not find the certificate for **msgspy** in **msgrecv**’s keystore. MQ AMS rejects the message and writes it to the **SYSTEM.PROTECTION.ERROR.QUEUE**. MQ AMS has prevented a rogue or unauthorized message from entering the network.

### Signed messages on local queue (with signature DN checklist)

This scenario is similar to the previous one. **msgsend** puts a message on queue **ORDERQ** and **msgrecv** receives that message. A policy is defined to use quality of protection **Integrity** (messages are signed). But in this case, a signature DN checklist is also provided which includes the DN of **msgsend** only. The userID **msgsend**\*\*\*\* then puts its own message on queue **ORDERQ** and subsequently **msgrecv** attempts to retrieve **msgsend**’s message. The policy for queue ORDERQ must be modified to add the signature DN checklist. User msgsend only is allowed to sign messages.

1. Modify the protection policy for queue ORDERQ to add the signature DN checklist.
   1. From userID **ibmdemo**, enter the setmqspl command.

* setmqspl -m QMAMS -p ORDERQ -s SHA1 -a “CN=msgsend,O=ibm,C=US”
* b. Display the policy you just modified. From userID **ibmdemo**:
* dspmqspl -m QMAMS -p ORDERQ
* You can also use the MQ Explorer, and display (or modify) the Protection Policies for the ORDERQ. This extension was added to your MQ Explorer with the installation of WMQ-AMS, and can be seen under the Advanced folder within any AMS-enabled queue manager.

1. User msgsend puts a message on queue ORDERQ.
   1. Change to the msgsend command window.
   2. As **msgsend**, use the sample program amqsput to put a message on ORDERQ.

* amqsput ORDERQ QMAMS
* This is another test message from msgsend

1. User **msgrecv** retrieves the message with no problem.
   1. Change to the msgrecv command window.
   2. As **msgrecv**, use the sample program amqsget to get a message from ORDERQ.

* amqsget ORDERQ QMAMS

1. Now try sending from a userID that isn’t AMS authorized. User msgspy puts a message on queue ORDERQ.
   1. Change to the msgspy command window.
   2. As **msgspy**, use the sample program amqsput to put a message on ORDERQ.

* amqsput ORDERQ QMAMS
* This is a test message from msgspy on ORDERQ

1. User msgrecv attempts to get the message from queue ORDERQ.
   1. Return to msgrecv’s command window.
   2. Use the sample program amqsget to get a message from ORDERQ.

* amqsget ORDERQ QMAMS
* **msgrecv** is unable to retrieve msgspy’s ’s message because, based on the policy defined, only msgsend’s message can be signed. msgspy’s message is sent to the error queue.

1. You may browse the **SYSTEM.PROTECTION.ERROR.QUEUE** queue to see the message.
   1. Return to **ibmdemo**’s command window.
   2. Use the sample program **amqsbcg** to browse the last message on the **SYSTEM.PROTECTION.ERROR.QUEUE**.

* amqsbcg SYSTEM.PROTECTION.ERROR.QUEUE QMAMS
  1. Review the message beginning with the header. Notice the format of MQDEAD and Useridentifier. Continue to browse the rest of the message.

1. When WMQ-AMS detects a security issue with messages, it logs error and warning messages in the standard WMQ queue manager error logs. You will now go and look at QMAMS’s error log to see some of those messages.
   1. Using the Windows Explorer, navigate to directory C:and Check the log file.
   2. Using Windows Explorer, navigate to the following directory and open the first error log file check the error in the error log file - **AMQERR01**.

* ### Signed and encrypted messages on local queue

In this scenario, **msgsend** puts a message on queue **ORDERQ**. The message is signed and encrypted for user **msgrecv**. A policy of type **Privacy** is created with msgrecv being the only recipient. Only msgrecv can retrieve and decrypt the message. Any other users attempting to get the message from the queue will fail and the message will be placed on the error queue.

1. Modify the existing policy for queue ORDERQ to add encryption. The signer checklist is removed for this scenario. msgrecv is added as the only recipient. Note that at least one recipient must be specified when encryption is requested.
   1. As user **ibmdemo**, issue the following command:

* setmqspl -m QMAMS -p ORDERQ -s SHA1 -e 3DES -r CN=msgrecv,O=ibm,C=US
  1. Display the policy you just updated.
* dspmqspl -m QMAMS -p ORDERQ
* Alternatively, you can also display (or modify) the Protection Policy using the MQ Explorer plugin.

1. When encryption is requested, the symmetric key generated is encrypted with the public key of the recipient (user **msgrecv** in this example). This requires user **msgsend** to have user **msgrecv**’s public certificate in his keystore. Because the AMS encryption option also implies integrity checking, user **msgsend**’s public key must also be accessible in user **msgrecv**’s keystores (CMS and JKS). This was done as part of the lab set-up. The public key of each user has previously been added to each other’s key-store.
   1. Change to the msgsend command window.
   2. As *msgsend*, use the sample program **amqsput** to put a message on **ORDERQ**:

* amqsput ORDERQ QMAMS
* This is a signed and encrypted test message

1. You will now browse the **ORDERQ\_AL** queue as user **ibmdemo** to verify that the message is encrypted. Remember that ORDER\_AL is an alias queue for **ORDERQ**. The alias is not protected and has been setup to demo encryption in this lab. This is not a best practice and should not be done in the real world (at least not without proper authorization).
   1. From the **ibmdemo** command window use the sample program amqsbcg to browse.

* amqsbcg ORDERQ\_AL QMAMS
* If there is more than one message on the queue, let it scroll all the way to the end, to make sure you are viewing the last message on the queue. Notice that the message contents are no longer readable. The message is indeed encrypted.
* (NB You could also have browsed the message using WMQ Explorer or RFHUtil, as before.)

1. User msgrecv retrieves the encrypted message with no problem.
   1. Change to the msgrecv command window.
   2. As **msgrecv**, use the sample program amqsget to get a message from **ORDERQ**.

* amqsget ORDERQ QMAMS

1. Put another new encrypted message from **msgsend**.
2. User **msgspy** attempts to retrieve the message from queue ORDERQ
   1. Change to the msgspy command window.
   2. As **msgspy**, use the sample program amqsget to get a message from ORDERQ.

* amqsget ORDERQ QMAMS
* **msgspy** is unable to retrieve **msgsend**’s message because, based on the policy defined, the only recipient who can retrieve and decrypt the message is **msgrecv**. The message is again automatically sent to the error queue by the AMS interception code.

1. You may browse the **SYSTEM.PROTECTION.ERROR.QUEUE** queue to see the message.
   1. Return to ibmdemo’s command window.
   2. Use the sample program **amqsbcg** to browse the message on the **SYSTEM.PROTECTION.ERROR.QUEUE**.

* amqsbcg SYSTEM.PROTECTION.ERROR.QUEUE QMAMS
  1. Allow the program to scroll down to the last message.
* Review the message beginning with the header. Notice the format of MQDEAD and user identifier. Continue to browse the rest of the message. You will see nothing but encrypted text in the message body. The message remains encrypted when put the SYSTEM.PROTECTION.ERROR.QUEUE.

1. Check the log file.
   1. Using Windows Explorer, navigate to directory **C:\* and open the first error log -** AMQERR01\*\* with your favorite editor.
   2. Move to the bottom of the file and you should see messages like those shown below.

* Of course, your file name and time/date will differ from the screen capture.

### Signed and encrypted messages on remote queue

In this scenario, **msgsend** puts a message on queue **REMOTE\_ORDERQ** on IIBQMGR and the message flows to queue **ORDERQ** on **QMAMS**. The message is signed and encrypted for user **msgrecv** using the same policy defined in the previous scenario. Only msgrecv can retrieve and decrypt the message. Any other users attempting to get the message from the queue will fail and the message will be placed on the error queue. The remote queue **REMOTE\_ORDERQ** on **IIBQMGR** needs the same policy as its local queue ORDERQ on QMAMS. So, because we have recently changed the policy on **ORDERQ**, we now need to change the policy on **REMOTE\_ORDERQ**.

1. Define the policy for queue **REMOTE\_ORDERQ** on **IIBQMGR**.
   1. As user **ibmdemo**, issue the following command:

* setmqspl -m IIBQMGR -p REMOTE\_ORDERQ -s SHA1 -e 3DES -r “CN=msgrecv,O=ibm,C=US”
  1. Display the policy you just created with the following command:
* dspmqspl -m IIBQMGR -p REMOTE\_ORDERQ
* Again, alternatively you could Define and Display the Protection Policy using the MQExplorer plugin. 2. Once again, you need to provide the necessary MQ access authorities to any users on the **IIBQMGR**. Authorize user **msgsend** to connect to MQ, retrieve policies and PUT messages to this protected queue. a. As user **ibmdemo**, issue the following commands:
* setmqaut -m IIBQMGR -t qmgr -p msgsend +connect +inq setmqaut -m IIBQMGR -t q -n SYSTEM.PROTECTION.POLICY.QUEUE -p msgsend +browse setmqaut -m IIBQMGR -t q -n REMOTE\_ORDERQ -p msgsend +put
* 3. User msgsend puts a message on queue **REMOTE\_ORDERQ**.
  1. Return to the **msgsend** command window.
  2. As **msgsend**, use the sample program amqsput to put a message on **REMOTE\_ORDERQ** on queue manager **IIBQMGR**.
* amqsput REMOTE\_ORDERQ IIBQMGR
* This is a test message from REMOTE\_ORDERQ
* 4. User **msgrecv** retrieves the encrypted message with no problem.
  1. Return to **msgrecv** command window.
  2. As **msgrecv**, use the sample program amqsget to get the message from **ORDERQ**. amqsget ORDERQ QMAMS
* !### Signed and encrypted messages on cluster queue In this scenario, **msgsend** puts a message to cluster queue **CLUSTER\_ORDERQ** from queue manager **IIBQMGR** and **msgrecv** retrieves the message from the queue on queue manager **QMAMS**. Note that the policy for queue CLUSTER\_ORDERQ must be defined on both queue managers, even though the queue itself may only exist on one queue manager.

1. Set the necessary MQ authorizations.
   1. As user **ibmdemo**, issue the following commands:

* setmqaut -m IIBQMGR -t q -n SYSTEM.CLUSTER.TRANSMIT.QUEUE -p msgsend +put
* setmqaut -m QMAMS -t q -n SYSTEM.CLUSTER.TRANSMIT.QUEUE -p msgsend +put
* setmqaut -m QMAMS -t q -n CLUSTER\_ORDERQ –p msgrecv +get
* setmqaut -m QMAMS -t q -n CLUSTER\_ORDERQ –p msgsend +put
  1. The following commands are also necessary for this scenario because the MQ cluster distribution may send the message to the instance of the queue on IIBQMGR, in which case you may need to get the message with the userID msgrecv. The following commands are all necessary to authorize msgrecv to connect to IIBQMGR and get a message. Issue the commands: setmqaut -m IIBQMGR -t qmgr –p msgrecv –p msgsend +connect +inq ``` setmqaut -m IIBQMGR -t q -n CLUSTER\_ORDERQ –p msgrecv +get
* setmqaut –m IIBQMGR –t q –n SYSTEM.PROTECTION.POLICY.QUEUE –p msgrecv –p msgsend +browse
* 2. Create the policies on both queue managers.
  1. As user **ibmdemo**, issue the following commands: ``` setmqspl -m QMAMS -p CLUSTER\_ORDERQ -s SHA1
* setmqspl -m IIBQMGR -p CLUSTER\_ORDERQ -s SHA1
* 3. User **msgsend** puts a message on queue *CLUSTER\_ORDERQ*.
  1. Return to msgsend‘s command window.
  2. As **msgsend**, use the sample program amqsput to put a message onto ORDERQ. ``` amqsput CLUSTER\_ORDERQ QMAMS
* This is a test message to CLUSTER\_ORDERQ

1. User **msgrecv** retrieves the signed message with no problem.
   1. Return to msgrecv’s command window.
   2. As msgrecv, use the sample program amqsget to get the message from *CLUSTER\_ORDERQ* on IIBQMGR.  
      amqsget CLUSTER\_ORDERQ IIBQMGR
   3. If there are no messages on the CLUSTER\_ORDERQ on queue manager IIBQMGR, run the sample program amqsput from msgsend again specifying queue manager QMAMS. Since the default cluster workload distribution is round-robin, every other message should go to CLUSTER\_ORDERQ on IIBQMGR. ## CONGRATULATIONS! ### You have completed this hands-on lab.For extra practice, you may continue with Java interceptors described below if time permits, or later on your own. ### Using other types of interceptors If time permits, you may want to go through the previous scenarios again, but this time use another type of interceptor (C client interceptor or the Java interceptors) using amqsputc/amqsgetc and other Java samples. When using MQ Java or JMS applications in bindings mode, two different interceptors can be used:

* Server interceptor (API exit)
  + The server interceptor will also handle Java applications connecting to a queue manager in bindings mode.
  + Java interceptor
  + To use the Java interceptor to protect messages of Java applications connecting to a queue manager in bindings mode, the following additional option must be passed on the Java command line:
  + -Dmqs.intercept.bindings=1
  + \* By default, the Java interceptor only handles client connections. Be aware that if both are enabled at the same time, then the cryptographic operations will be performed twice!